DIGITAL NETWORK THEORY AND ITS APPLICATION TO THE ANALYSIS AND DESIGN OF DIGITAL FILTERS

by

Ronald Eldon Crochiere

S.B., Milwaukee School of Engineering
(1967)

S.M., Massachusetts Institute of Technology
(1968)

E.E., Massachusetts Institute of Technology
(1972)

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

April, 1974

Signature of Author _____
Department of Electrical Engineering, April 30, 1974

Certified by _____
Thesis Supervisor

Accepted by _____
Chairman, Departmental Committee on Graduate Students

DIGITAL NETWORK THEORY AND ITS APPLICATION TO THE ANALYSIS AND DESIGN OF DIGITAL FILTERS

by

Ronald Eldon Crochiere

Submitted to the Department of Electrical Engineering on April 30, 1974 in partial fulfillment of the requirements for the Degree of Doctor of Philosophy.

ABSTRACT

In this dissertation we investigate the problems of analysis and design of efficient digital filter structures with particular emphasis on coefficient word length and parallelism in their implementations. A general network theory formalism for digital networks is proposed and a matrix representation is presented which uniquely characterizes a structure. This network theory formalism then serves as a basis for proving a number of general theorems and coefficient sensitivity properties of digital networks. It also serves as a basis for the development of efficient techniques for computer-aided analysis of arbitrary digital networks. A computer-aided digital network analysis package (CADNAP) is developed as part of this thesis using these techniques.

In a second phase of the thesis emphasis is placed on minimizing the coefficient word length of a digital filter. A statistical measure for estimating the required word length is presented. An optimization technique is then proposed for choosing the filter approximation such that this statistical word length measure is minimized for a given structure and a given set of filter specifications. It is demonstrated by example that this technique can lead to a reduction in the required word length by one to three bits. Finally, in an effort to develop intuition into what types of filter structures may have low coefficient word length a number of different structures are analyzed and compared for a specific example. The analyses of these structures were performed using CADNAP.

THESIS SUPERVISOR: Alan V. Oppenheim
TITLE: Associate Professor of Electrical Engineering

ACKNOWLEDGEMENT

I wish to express my gratitude to my advisor Professor A. V. Oppenheim for his guidance, encouragement, and suggestions throughout the course of this thesis. Working with him has been a great privilege and experience. Also I am greatly indebted to Professor P. L. Penfield, Jr. and Doctor J. F. Kaiser for their many discussions and suggestions.

I would like to express my sincere appreciation to my wife Gail and also to our families for their patience, assistance, and encouragement during my graduate study.

I wish to acknowledge support from the National Science Foundation both in the form of a traineeship and as a research assistantship (Grant GK-31353).

Finally I wish to express thanks to my wife for her assistance in typing and preparing the proposal and the first draft of the thesis and to Monica Pettijohn for her excellent job of typing the final manuscript.

TABLE OF CONTENTS

Chapter I

INTRODUCTION AND SCOPE OF THE THESIS

1.1 OUTLINE OF PROBLEM AREA

The field of digital filter design for digital signal processing applications has received considerable attention in recent years [1], [2], [3], [4], [73], [74]. There has been an increasing interest in the use of special purpose digital hardware to perform digital filtering operations. It is often desired in such applications, for the sake of economy, to keep the word lengths for the filter coefficients and signals to a minimum and to utilize computational algorithms which can be efficiently realized with a minimum of hardware. If speed is an important requirement we may be interested in algorithms which can be performed with a maximum amount of parallelism in hardware. In both cases an important consideration in the design is the structure by which the filter is implemented. In response to this problem many types of filter structures have been recently proposed, by various authors, as candidates for efficient filter design and inevitably more different types of structures will be proposed in the future.

In a particular digital filter application there is the problem of comparing these various candidates and choosing the one which best meets the requirements of that application. To do this a set of criteria are needed for the comparison and in

the process of doing a comparison a considerable amount of analysis and simulation of some of these structures may have to be performed. Once a particular structure is chosen we may then ask how we can best match the design problem to the characteristics of this structure. It is these kinds of problems which we have addressed ourselves to in this dissertation. Particular emphasis is placed on the problems of analysis and comparison of structures with respect to coefficient word length, parallelism, and efficiency of implementation. Emphasis is also placed on the problem of minimizing the coefficient word length of a particular structure when a set of design specifications for a particular application are known.

1.2 REVIEW OF PREVIOUS WORK

A considerable amount of effort has been put forth by numerous authors on various aspects of synthesis and comparison of digital filter structures [4], [73]. It would not be possible to discuss in a short review the work of all of those who have been involved. Consequently we will only attempt to mention that work which is directly related to this thesis or is most familiar to this author.

Some of the earlier efforts in the design and comparison of digital filter structures are well documented in the works of Kaiser and Kuo [1], [72] and Gold and Rader [2]. Considerable efforts in the direction of analyzing, synthesizing, and comparing digital filter structures have since been made by

Jackson and Kaiser [24], [32], [69], Oppenheim and Weinstein [3], [70], Knowles and Olcayto [50], Schüssler [71], and many others [4], [73], [74].

Recently Avenhaus [57] has extended some of the concepts of Knowles and Olcayto in using statistical methods for comparing filter structures on the basis of coefficient word length. This work has led to the concept of a statistical measure of the coefficient word length. Similar techniques have been applied by Chan and Rabiner [68] in the analysis of finite impulse response filters.

In other directions, general concepts of digital network properties and theorems such as Tellegen's theorem for digital networks and first-order coefficient sensitivity relations for digital networks have been proposed by Seviora and Sablatash [20], Fettweis [21], and more recently by Lee [22], [23].

In the area of new types of filter structures, considerable efforts have recently been made by many authors. Avenhaus [34] and Parker and Hess [33] have proposed many new forms of second-order filter sections for cascade or parallel filter structures. Mitra and Sherwood [18], [31] have proposed a number of new types of structures which can be synthesized with the aid of continued fraction expansions and two port network theory concepts. Fettweis [15], [16], [58], [66] has proposed a class of digital filter structures which can be synthesized from, and imitate the properties of classical analog filter circuits. Gray and Markel [35] have proposed

a ladder structure which is related to the concepts of vocal tract modeling in speech research. We can perhaps expect that many more different classes of structures will be proposed in the future.

1.3 OUTLINE AND RESULTS OF THE THESIS

The thesis can be divided into four major parts. The first part, covered in Chapters 2, 3, and 4, involves the development of a general network theory formalism for digital networks. This formalism is based on an extension of the concepts of signal-flow graphs by Mason [5], [6]. A matrix representation is proposed and serves as a mathematical framework for this formalism. In Chapter 3 a number of higher-order and large-change coefficient sensitivity relations are developed by extending the first-order network sensitivity results of Fettweis [21] and Seviora and Sablatash [20]. Also a connection between group delay and specific coefficient sensitivities in a structure is established using a property of homogeneous functions. In Chapter 4 concepts of precedence, parallelism, serialism, and pipelining are discussed in the context of this network theory framework.

These network properties and the matrix representation are applied in the second part of the thesis to the development of general computer-aided network analysis techniques for digital networks. As a part of the thesis a computer-aided digital network analysis package (CADNAP) is developed. These

concepts are presented in Chapter 5 and the documentation and programs for CADNAP are given in the appendices.

The third part of the thesis, covered in Chapter 6, is concerned with the problem of minimizing the coefficient word length when a filter structure and a specific set of filter specifications are given. The statistical word length concepts of Avenhaus [57] are extended and it is shown experimentally that the computations involved in determining statistical word lengths for equiripple filters can be performed very efficiently. An optimization procedure is proposed for reducing the coefficient word length, based on minimizing the statistical word length. Improvements of one to three bits are observed experimentally using this procedure.

The fourth topic is treated in Chapters 4 and 7. In Chapter 4 a number of recently proposed structures are compared on the basis of the amount of parallelism and serialism contained in these structures and on the number of operations that must be performed to implement these structures for arbitrary system functions. In Chapter 7 a comparison is made for a specific bandpass, elliptic filter example. The comparison is made on the basis of the statistical coefficient word length measure. The analyses of these structures were made with the use of the CADNAP programs.

Chapter II

NETWORK THEORY OF DIGITAL FILTERS

2.1 INTRODUCTION

The study of the properties of digital filter structures and the principles of analysis and synthesis of these structures inherently involves the fundamental concepts of network theory. In this chapter these concepts, as they apply to digital networks, are formulated and a general network representation for digital networks is presented.

Digital networks can be placed in the category of directed graph theory or signal-flow graphs [5], [6], [7]. Consequently, many of the theorems and methods of signal-flow graphs can be specifically adapted to digital networks. These concepts serve as the basis for the general network representation.

2.2 SIGNAL-FLOW GRAPH REPRESENTATION OF DIGITAL NETWORKS

A signal-flow graph is a finite collection of nodes and directed branches which graphically depict the order of signal flows and the operations on these signals within a network. Associated with each node n is a node signal value, $y_n(k)$, and associated with each branch is a branch signal. For the case of digital networks these signals are discrete-time signals with k corresponding to the "time index".

A regular branch of the network is defined to be a directed branch which exits from some node m and enters some node n, where m and n may correspond to the same node or to two different nodes within the network (see Fig. 2.1). Node m is referred to as the exit node of the branch and node n is referred to as the entrance node of the branch. Associated with these nodes are the node signal values $y_m(k)$ and $y_n(k)$ respectively. Each regular branch has an input signal which is the node signal value of its exit node and an output signal which will be denoted as the branch signal value, $w(k)$. The branch signal value corresponds to the result of the branch transformation, $F[\cdot]$, on its input. That is,

$$w(k) = F[y_m(k)]. \tag{2.1}$$

It will be assumed, unless otherwise specified, that for all of the regular branches in the network the branch transformations are causal, linear, and shift-invariant. The branch transformation, $F[\cdot]$, can then be characterized by its unit sample response, $f(n)$, and its z transform, $F(z)$, where $F(z)$ will be defined as the branch transmittance function of the branch. The input/output relation of a regular branch can then be specified by the transform relation

$$W(z) = F(z)Y_m(z), \tag{2.2}$$

where $W(z)$ and $Y_m(z)$ correspond to the z transforms of $w(k)$

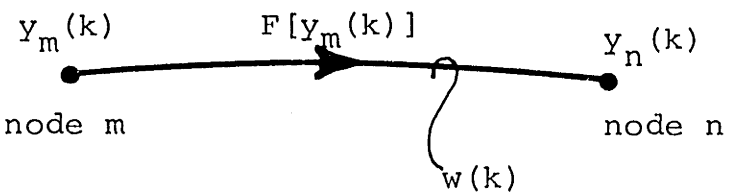


Fig. 2.1 Definition of a regular branch.

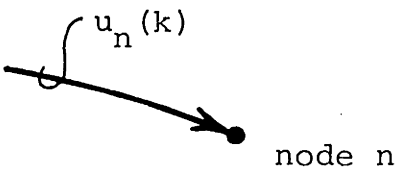


Fig. 2.2 Definition of a source branch.

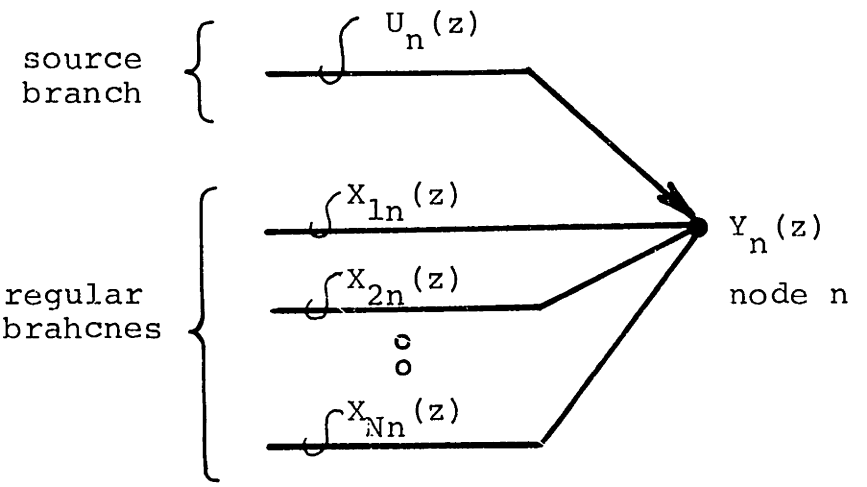


Fig. 2.3 Definition of a node

and y(k), respectively.

In general, there may be more than one branch directed from node m to node n in the network. In this case, the total branch signal of branches directed from node m to node n, denoted as $x_{mn}(k)$, is defined as the sum of the branch signal values of all of the regular branches in the network which are directed from node m to node n. That is,

$$x_{mn}(k) = \sum_{\substack{\forall \text{ Branches} \\ m \to n}} w(k) \quad , \tag{2.3}$$

where the notation under the summation is meant to read "the sum over all regular branches directed from node m to node n". By taking the z transform of (2.3) and applying (2.2), we get

$$X_{mn}(z) = H_{mn}(z)\ Y_m(z), \tag{2.4a}$$

where

$$H_{mn}(z) = \sum_{\substack{\forall \text{ Branches} \\ m \to n}} F(z) \quad . \tag{2.4b}$$

In the above relation $H_{mn}(z)$, denoted as the total branch transmittance function from node m to node n, is defined as the sum of the branch transmittance functions of all regular branches in the network which are directed from node m to node n. If no branches exist from node m to node n, it is

equivalent to saying that $H_{mn}(z) = 0$; which, in turn, implies that $X_{mn}(z) = 0$.

A second type of branch which we consider in the flow graph representation is a source branch (see Fig. 2.2). A source branch is a branch which represents the injection of an external signal into the graph or network at some node n. It has an entrance node, n, but does not have an exit node or a branch transmission function associated with it. The branch signal value of the source branch, denoted as $u_n(k)$, corresponds to the sum of all of the external signals injected at node n. It will be assumed without any loss in generality that each node in the network can have only one source branch entering it. If no external sources are injected in that node it is equivalent to saying that the source branch associated with that node has a branch signal value equal to zero. A graph which has no source branches or equivalently, all of its source branches have zero branch signal values, will be defined as a homogeneous graph or an undriven network and corresponds to a digital network with no inputs.

The node signal value, $y_n(k)$ or $Y_n(z)$, of a node, n, can now be defined as the sum of the branch signal values of all branches entering node n (see Fig. 2.3). In effect, the node performs the function of an adder in that it sums all of the branch signals entering it. It will be assumed that if no branch exists from some node m to node n, $X_{mn}(z) = H_{mn}(z) = 0$, and if no source branch enters node n, $U_n(z) = 0$. It will be

further assumed that the total number of nodes in the network is equal to N and that these nodes are numbered consecutively from 1 to N. With these assumptions, the node signal value, $Y_n(z)$, of node n can be expressed as

$$Y_n(z) = U_n(z) + \sum_{m=1}^{N} X_{mn}(z). \tag{2.5}$$

Alternatively, using (2.4a), this can be written in the form:

$$Y_n(z) = U_n(z) + \sum_{m=1}^{N} H_{mn}(z)\, Y_m(z). \tag{2.6}$$

As there are N nodes in the network, there are N equations of the form of (2.5) or (2.6) for the network corresponding to n equals 1 through N. These N equations provide a complete description of the signal-flow graph.

The above definitions for the node, regular branch, and source branch provide all of the essential elements necessary to describe a digital network. To illustrate the use of signal-flow graphs for representing a digital network, an example of a cascade form digital filter with 3 poles and 3 zeros is given in Fig. 2.4. Branch transmittances are written beside the arrow of the branch. If no value is given for a particular branch, it is assumed that its transmittance is unity.

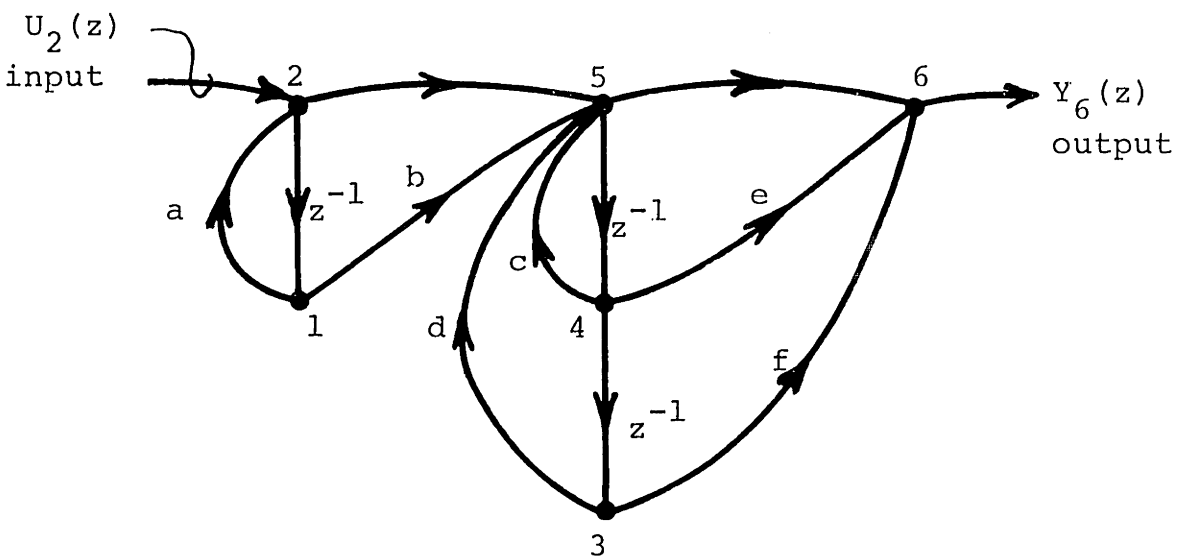


Fig. 2.4 Example of a signal-flow graph representation of a digital network.

2.3 A MATRIX FORMULATION OF THE NETWORK EQUATIONS

The set of N equations of the form of (2.6) for n = 1 to N provide a total description of the digital network. They can be written more compactly in matrix form [7] as

$$\underline{Y}(z) = \underline{U}(z) + \underline{H}^{t}(z)\ \underline{Y}(z), \tag{2.7}$$

where

$\underline{Y}(z)$ = a column vector of the node signal values $Y_n(z)$,

$\underline{U}(z)$ = a column vector of the source branch signals $U_n(z)$,

and

$\underline{H}(z)$ = the N × N matrix of the total branch transmittance functions $H_{mn}(z)$.

The superscript t on the matrix $\underline{H}^{t}(z)$ represents the matrix transpose. This is necessary in equation (2.7) because of a conflict between the conventional subscript notation used in signal-flow graph theory and the conventional subscript notation used in matrix theory.

A classical problem in system theory is that; given a network and one or more inputs, it is desired to determine one or more outputs or node signal values due to these inputs. With the matrix formulation given by (2.7) this problem reduces to one of determining a matrix inverse. Specifically,

$$(\underline{I} - \underline{H}^{t}(z))\ \underline{Y}(z) = \underline{U}(z)$$

$$\underline{Y}(z) = \underline{T}^{t}(z)\ \underline{U}(z) \tag{2.8}$$

where

$$\underline{T}(z) = [(\underline{I} - \underline{H}^{t}(z))^{-1}]^{t} = (\underline{I} - \underline{H}(z))^{-1}. \qquad (2.9)$$

The matrix $\underline{T}(z)$ is referred to as the transfer function matrix and $T_{mn}(z)$, the $(m,n)^{th}$ element of $\underline{T}(z)$, is the transfer function from node m to node n. Written for a single node, n, (2.8) becomes:

$$Y_n(z) = \sum_{m=1}^{N} T_{mn}(z)\ U_m(z). \qquad (2.10)$$

For the case of a network or a graph in which a specific source $U_i(z)$ is considered as the input and a specific node value $Y_j(z)$ is considered as the output, the particular transfer function $T_{ij}(z)$ is referred to as the system function of the graph or network.

Matrix inversion is not the only way to determine the system function of a graph or network. Other conventional methods include Cramer's rule, solution by linear equation solving (e.g. Gaussian elimination) [8], [9], Mason's gain formula [5], [6], and graph reduction techniques [5], [6], [10]. The last two methods are particularly suited for hand computation of relatively small networks.

2.4 ELEMENTARY DIGITAL NETWORKS AND THEIR MATRIX REPRESENTATION

In its elementary form, a digital structure is a

pictorial representation of a set of first-order difference equations. To implement the filter by computer or with special purpose hardware, only three basic functions are needed to compute these difference equations. They consist of multiplying a signal by a gain constant, delaying a signal by a unit delay, and adding one or more signals together. In a signal-flow graph representation of a digital structure in its elementary form, only two types of branches are needed to represent these basic operations (see Fig. 2.5). The first type of branch corresponds to a constant gain for which the branch transmittance is equal to a constant

$$F(z) = c. \tag{2.11}$$

The second type of branch corresponds to a gain in cascade with a unit delay for which the branch transmittance is

$$F(z) = dz^{-1}. \tag{2.12}$$

The gain constant d can be assigned to be unity for all delay branches without any loss in generality; however, this assumption will not be required unless otherwise specified.

A further restriction which will be made in the representation of a digital network in its elementary form, without any loss in generality, is that only one branch of a kind (coefficient or coefficient and delay branch) will be allowed to exist from one node to another. That is, two or more branches of the same kind in parallel (in the same direction)

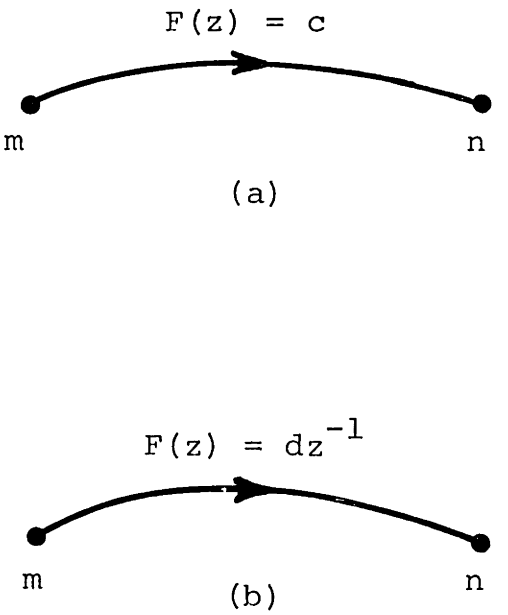


Fig. 2.5 Allowed branches for an elementary network, (a) gain and (b) gain and delay.

between any two nodes are not allowed. A flow graph representation of a digital network which satisfies the above conditions will be defined as an elementary network. As an example, the network in Fig. 2.4 corresponds to an elementary network. All other network representations which do not satisfy these restrictions will be referred to as nonelementary networks. An example of a nonelementary network might be one in which the branches correspond to more complex functions such as a bandpass filter.

If a network satisfies the above conditions which define an elementary network, then $H_{mn}(z)$ will have two terms, one corresponding to a coefficient branch and one corresponding to a coefficient and delay branch. These two terms will be represented respectively as h_{mnc} and $h_{mnd}z^{-1}$, where h_{mnc} and h_{mnd} correspond to the constant coefficients of the respective branches. This applies for all combinations of m and n in the network. Therefore, if a network is an elementary network, then its matrix representation, (2.7), can be expressed in the form

$$\underline{Y}(z) = z^{-1}\, \underline{H}_d^t\, \underline{Y}(z) + \underline{H}_c^t\, \underline{Y}(z) + \underline{U}(z), \qquad (2.13)$$

where

$\underline{Y}(z)$ = the column vector of node signal values,

$\underline{U}(z)$ = the column vector of source branch values,

$\underline{H}_c$ = the matrix of coefficients for branches with simple coefficients,

and

$\underline{H}_d$ = the matrix of coefficients for branches with coefficients and delays.

In this representation all of the coefficients and all of the signal values in the digital network are uniquely expressed. That is, there exists a unique one-to-one relationship between the network and its matrix representation in the form of (2.13). Given the matrix equation of a network, one can completely reconstruct the network in graphical form. Where no branches occur in the network, the corresponding elements of $\underline{H}_c^t$ or $\underline{H}_d^t$ are equal to zero.

The corresponding matrix of transfer functions for (2.13) can be expressed as

$$\underline{T}^t(z) = (\underline{I} - \underline{H}_c^t - \underline{H}_d^t z^{-1})^{-1}. \tag{2.14}$$

The matrices $\underline{H}_c$ and $\underline{H}_d$ are matrices of constant coefficients which correspond to the coefficients in the network. Therefore, the inverse z transform of (2.13) can be taken to obtain a matrix equation corresponding to N first-order difference equations. This equation can be expressed as

$$\underline{y}(k) = \underline{H}_d^t \underline{y}(k-1) + \underline{H}_c^t \underline{y}(k) + \underline{u}(k). \tag{2.15}$$

It is this set of equations which is iterated in the "time domain" by computer or with special purpose hardware in the actual operation of the filter.

For the example of Fig. 2.4 the set of equations of (2.15) take the form

$$\begin{bmatrix} y_1(k) \\ y_2(k) \\ y_3(k) \\ y_4(k) \\ y_5(k) \\ y_6(k) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} y_1(k-1) \\ y_2(k-1) \\ y_3(k-1) \\ y_4(k-1) \\ y_5(k-1) \\ y_6(k-1) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ a & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ b & 1 & d & c & 0 & 0 \\ 0 & 0 & f & e & 1 & 0 \end{bmatrix} \begin{bmatrix} y_1(k) \\ y_2(k) \\ y_3(k) \\ y_4(k) \\ y_5(k) \\ y_6(k) \end{bmatrix} + \begin{bmatrix} 0 \\ u_2(k) \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{2.16}$$

2.5 ORDER OF COMPUTATION AND COMPUTABILITY OF ELEMENTARY DIGITAL NETWORKS

2.5.1 Order of Computation and Node Renumbering

So far no stipulation has been made as to the order in which the difference equations in (2.15) are iterated. It will be assumed in this chapter that this order will be computed from top to bottom. That is, the node values $y_i(k)$ will be computed from $i = 1$ to $i = N$, one at a time, in consecutive order in the actual implementation of the network. The input can be any one of the $u_i(k)$ sources and the output can be any one of the node values $y_i(k)$. In Chapter 4 we will consider the more general case where perhaps more than one node may be computed simultaneously.

The order in which the nodes are computed can be changed by renumbering them in the network. This is equivalent to changing the order of the scalar equations represented in the

matrix equation of (2.15). This operation can also be interpreted as a linear transformation of the matrix equation in (2.15) to a new matrix equation by premultiplying by an $N \times N$ permutation matrix $\underline{P}$. That is,

$$\underline{y}'(k) = \underline{P}\,\underline{y}(k)$$

$$= (\underline{P}\,\underline{H}_d^t\,\underline{P}^{-1})\,\underline{P}\,\underline{y}(k-1) + (\underline{P}\,\underline{H}_c^t\,\underline{P}^{-1})\,\underline{P}\,\underline{y}(k) + \underline{P}\,\underline{u}(k),$$

or,

$$\underline{y}'(k) = \underline{H}'^t_d\underline{y}'(k-1) + \underline{H}'^t_c\underline{y}'(k) + \underline{u}'(k), \qquad (2.17)$$

where the primed equation corresponds to the new matrix equation after the linear transformation. The permutation matrix $\underline{P}$ is a matrix which is obtained by appropriately interchanging rows or columns of the $N \times N$ identity matrix. It is always nonsingular. This new matrix equation represents the same digital network as before with the exception that the nodes are numbered in a different order.

2.5.2 Computability of Elementary Networks

Although it is possible to number the nodes in a network from 1 to N in any order desired, not all permutations of these orderings will permit the node signal values $y_i(k)$ to be computed in consecutive order; that is, for i = 1 through N, consecutively. For example, if there exists a coefficient branch with a nonzero coefficient from node j to node i, where

$j > i$, then $y_i(k)$ cannot be computed before $y_j(k)$ because the value of $y_j(k)$ is needed for computation of $y_i(k)$ (see Fig. 2.6). Consequently, a conflict arises and the computation cannot be performed in this particular order. An ordering of this type is said to be a noncomputable ordering. Another order for numbering the nodes must be chosen if the node variables are to be computed consecutively. This ordering is only dependent on the topology of the coefficient branches of the structure. Conflicts cannot arise from source branches because their signal values $\underline{u}(k)$ are assumed to be known at time index k. They also cannot arise from coefficient and delay branches as the inputs to these branches, $\underline{y}(k-1)$, are also assumed to be known at time index k either because they have been specified as initial conditions or because they were computed in the previous filter cycle at time index k-1.

In some cases, no ordering of the node numbers exists such that the node variables $\underline{y}(k)$ can be computed consecutively. If this occurs, the network is said to be noncomputable. If there is at least one way to number the nodes such that they can be computed consecutively at least one node at a time with no conflict, then the network is said to be computable. It is important to note that the fact that a network is noncomputable does not imply that the network equations cannot be solved. It means that they cannot be solved for each node variable successively according to the difference

Fig. 2.6 A conflict situation when computing node signal values in consecutive order.

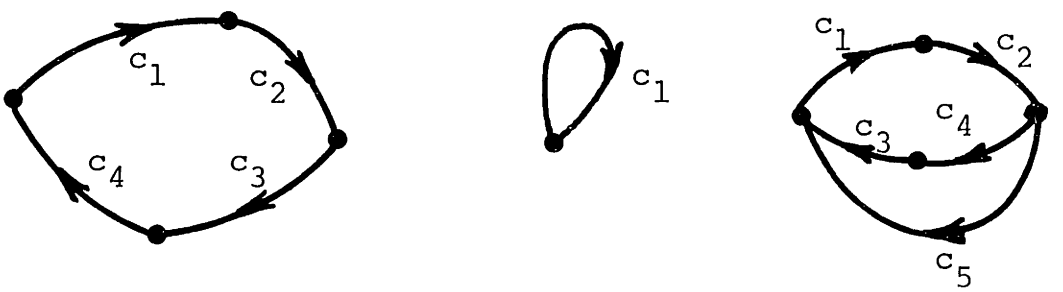


Fig. 2.7 Examples of closed loops without delay.

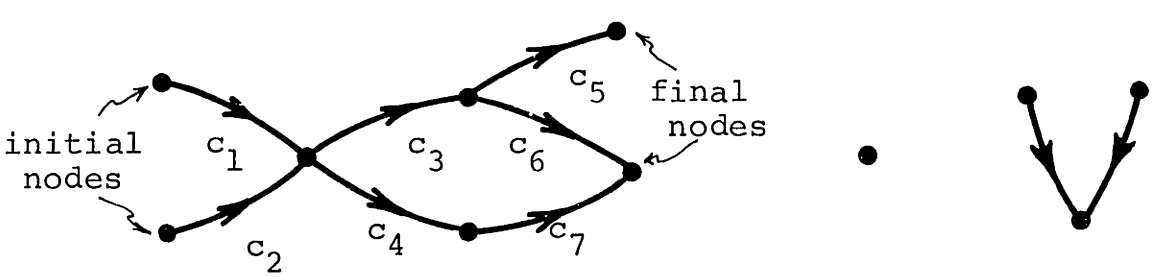


Fig. 2.8 Examples of chains.

equations in (2.15).

The property of computability of an elementary network is closely associated with the presence or absence of closed loops without delay in the network. As demonstrated above, this depends only on the topology of the coefficient branches in the network. To examine this property in more detail it is necessary to define some additional terms.

A closed loop without delay is defined as an interconnection of coefficient branches such that if we start at any node in the loop, it is possible to return to that node by following a connected, directed path along the coefficient branches (whose coefficients are nonzero) such that the direction of the path is always opposite to the branch directions. Alternatively, we can always find a second connected, directed path which is always in the same direction as the branches by choosing the path which is the reverse of the above path. Some examples of closed loops without delay are given in Fig. 2.7.

A chain is defined as an interconnection of coefficient branches (whose coefficients are nonzero) such that if we start at any node in the chain and follow any connected, directed path along the coefficient branches, the direction of which is always in the same direction of the branches or always in the opposite direction of the branches, it is not possible to return to the node from which we started. Associated with every chain in the network is at least one initial

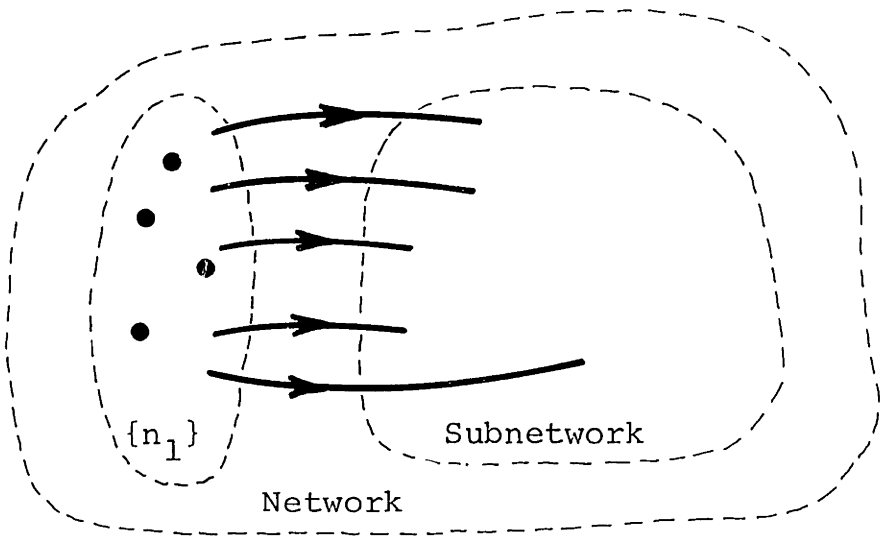
node and one final node, where an initial node is defined as a node for which there are no coefficient branches entering and a final node is defined as a node for which there are no coefficient branches leaving. This is true because there are only a finite number of nodes N in the network and a connected, directed path, as described above, can contain at most N nodes without closing upon itself. Therefore, all directed paths must be of finite length and have a beginning and an end if the structure is a chain. A node with no coefficient branches entering or leaving is considered as a chain in degenerate form such that its initial node and its final node are the same node. Some examples of chains are given in Fig. 2.8.

Lemma 2.1: If a network contains no closed loops without delay, then every node in the network is associated with a chain.

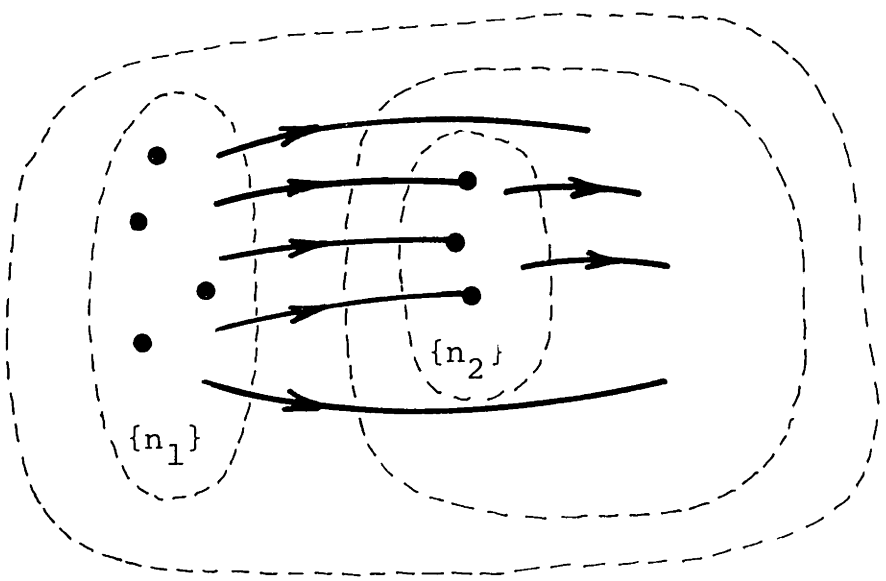
Proof: The lemma can be proved by observing that for every node in the structure we can start from that node and follow any possible directed, connected path through the coefficient branch structure (branches with zero coefficients are not allowed) such that the direction of the path is either always in the same direction as the branches or always in the opposite direction of the branches. Because there are no closed loops without delay in the network, these paths can never close upon themselves such that we can return to the node from which we started. Therefore, every node must be associated with a chain. Q.E.D.

Theorem 2.1: An elementary network is computable if and only if there are no closed loops without delay in the network.

Proof: The "if" part will be proved first. From Lemma 2.1, if there are no closed loops without delay then every node is associated with a chain. Furthermore, we know that every chain has at least one initial node. Consequently, the network has one or more initial nodes which will be denoted by the set of nodes $\{n_1\}$. As the initial nodes $\{n_1\}$ do not have coefficient branches entering them, all input signals to these nodes are known and therefore, their values can be computed. We can then consider the subnetwork which contains all of the nodes in the network except the initial nodes $\{n_1\}$ (see Fig. 2.9a). If we delete the nodes $\{n_1\}$ and the branches connecting nodes $\{n_1\}$ to the subnetwork, we can then treat the remaining subnetwork as a new network with its own set of initial nodes $\{n_2\}$ (see Fig. 2.9b). Each of the nodes in the set $\{n_2\}$ however, must be an entrance node for at least one coefficient branch which comes from one of the nodes in $\{n_1\}$ when considering the overall network. This can be shown by contradiction. If a node belongs to set $\{n_2\}$ then, by definition of an initial node, it does not have any coefficient branches entering it from the subnetwork. If it also does not have any coefficient branches entering it which come from the set $\{n_1\}$ then it has no coefficient branches entering it from any of the nodes in the network because the set of nodes $\{n_1\}$ and the nodes in the subnetwork taken together comprise all of



(a)



(b)

Fig. 2.9 (a) Extracting the initial nodes $\{n_1\}$ from a network and
(b) extracting the initial nodes $\{n_2\}$ from the subnetwork.

the nodes in the network. Consequently, the node is an initial node of the overall network and is contained in set $\{n_1\}$. As the two sets of nodes $\{n_1\}$ and $\{n_2\}$ cannot overlap, the node cannot simultaneously belong to both sets and, therefore, such a node cannot exist.

Once all of the node values of the initial nodes $\{n_1\}$ are computed then all of the inputs to the coefficient branches leading to the nodes $\{n_2\}$ are known and these nodes can be computed. We can now extend this process by considering subnetworks within the subnetwork (see Fig. 2.10). For each subnetwork i such that $i > 1$, there is at least one initial node in the subnetwork because there are no loops. All initial nodes of each subnetwork i are assigned to the initial node set $\{n_i\}$ for that subnetwork. Each subnetwork i is formed from the previous subnetwork i-1 by excluding the initial node set $\{n_{i-1}\}$ and its connecting branches from that subnetwork. The process is completed at stage f when all of the N nodes in the network have been assigned to initial node sets $\{n_i\}$ for $i = 1,2,3,\ldots,f$. Obviously, $f \leq N$, and all of the nodes in the final subnetwork are contained in the node set $\{n_f\}$. By the same argument used previously we can show by contradiction that each node in the set $\{n_i\}$ for $i > 1$ must be an entrance node of one or more coefficient branches which exit from the node set $\{n_{i-1}\}$. Also, each node in the set $\{n_i\}$ can be an entrance node for branches from any of the nodes in sets $\{n_j\}$ for $j = 1,2,\ldots,i-1$, but not from any nodes

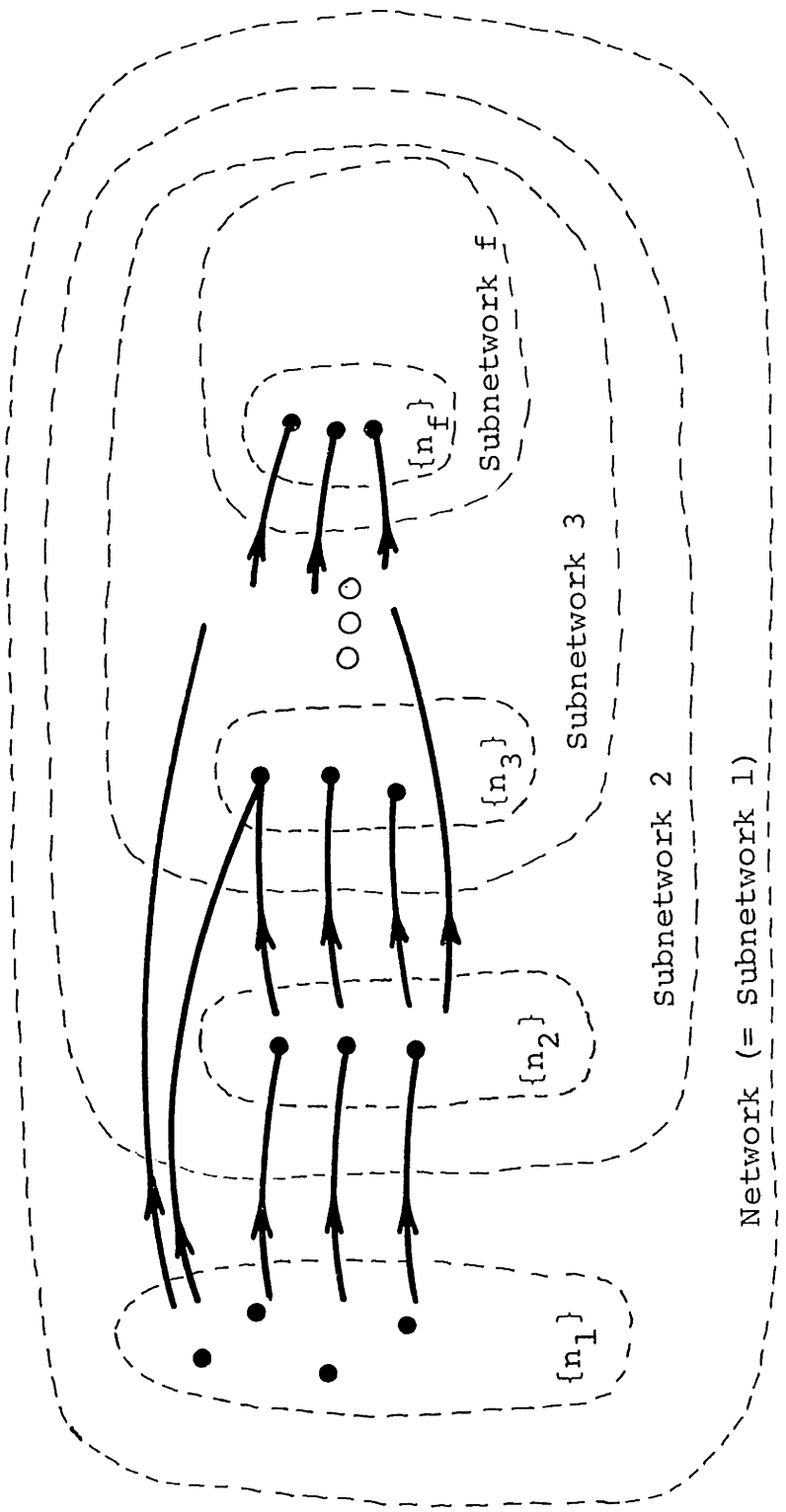


Fig. 2.10 Illustration of the successive breakdown of subnetworks of a computable network.

in the sets j = i,i+1,...,f. Therefore, once the node signal values for nodes in sets $\{n_j\}$ for j = 1,2,...,i-1 are known, the inputs to all coefficient branches entering nodes in $\{n_i\}$ are known and they can be computed. Furthermore, we showed that all of the nodes $\{n_1\}$ can be computed. Therefore, all of the nodes in the network can be computed without conflict by computing all of the nodes of set $\{n_1\}$, then all of the nodes of set $\{n_2\}$, and so forth for $\{n_3\}$, $\{n_4\}$,...,$\{n_f\}$. Thus, if there are no closed loops without delay in the network, it is computable. This completes the "if" part of the proof of Theorem 2.1.

The "only if" part of the proof can be demonstrated by showing that if there is a closed loop without delay in the network, then it is not computable. This can be seen by the fact that a connected, directed path on the loop taken to be always in the direction of the branches will by definition close upon itself. To compute each node in the loop, we need to know the node value of the previous node in the path. That is, there are no initial nodes in the loop. Therefore, no nodes in the loop can be computed. In the above algorithm for constructing a network in the form of Fig. 2.10, the procedure of extracting the initial nodes of each successive subnetwork terminates prematurely before using up all N nodes in the network because closed loops without delay do not have initial nodes. This completes the proof of Theorem 2.1.

Q.E.D.

Theorem 2.2: If and only if an elementary network contains no closed loops without delay, then the nodes in the network can be numbered in an order such that the matrix $\underline{H}_c^t$ in (2.15) is zero on and above the main diagonal as depicted in (2.18).

$$\underline{H}_c^t = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ h_{12} & 0 & 0 & & 0 \\ h_{13} & h_{23} & 0 & & 0 \\ \vdots & & & & \vdots \\ h_{1N} & \cdot & \cdot & \cdots & 0 \end{bmatrix} \qquad (2.18)$$

Proof: The "if" part of the proof can be shown by applying the results of the proof of Theorem 2.1. That is, if and only if the network contains no closed loops without delay, then it can be constructed in the form given in Fig. 2.10. From this form of the network, we can find an order for numbering the nodes such that $\underline{H}_c^t$ satisfies the above criterion. This can be accomplished by first numbering all of the nodes in set $\{n_1\}$ then $\{n_2\}$, $\{n_3\}$,..., $\{n_f\}$, consecutively until all of the nodes in the network have been numbered. That is, the nodes in set $\{n_1\}$ are numbered in any order from 1 through N_1 where N_1 is the total number of nodes in set $\{n_1\}$. Then for every node set $\{n_i\}$, $2 \leq i \leq f$, the nodes are numbered in any order from a through b where

$$a = 1 + \sum_{j=1}^{i-1} N_j,$$

$$b = \sum_{j=1}^{i} N_j,$$

and N_j is the total number of nodes in node set $\{n_j\}$.

For every element h_{mnc} in $\underline{H}_c^t$, if nodes m and n are in the same node set, then h_{mnc} must be zero because there are no coefficient branches interconnecting these nodes. If node m belongs to node set $\{n_i\}$ and node n belongs to node set $\{n_j\}$ then h_{mnc} must be zero if $i \geq j$ because of the construction of the network in Fig. 2.10. Furthermore, if the node numbers are chosen in the manner described above, then the condition $m \geq n$ implies that $i \geq j$ and, therefore, h_{mnc} must be zero for $m \geq n$. The elements h_{mnc} for $m \geq n$, however, correspond to elements in $\underline{H}_c^t$ which are either on or above the main diagonal. Therefore, the "if" part of the proof is completed.

The "only if" part of the proof can be demonstrated by proving the inverse of the theorem. That is, if the nodes in the network can be numbered in a manner such that $\underline{H}_c^t$ is zero on and above the main diagonal, then the network contains no closed loops without delay. From the converse of Theorem 2.1, we have shown that if a network is computable, then there are no closed loops without delay. Therefore, it is sufficient

to show that if the nodes in the network can be numbered in a way such that $\underline{H}_c^t$ is zero on and above the main diagonal, as in (2.18), then the network is computable. This can be established by recalling that in the computation of the elements $\underline{y}(k)$ in (2.15), all elements of $\underline{u}(k)$ and $\underline{y}(k-1)$ are known for that time index, k. The only sources of conflict which can occur are if the computation of some element $y_i(k)$ requires the value an element $y_j(k)$ which has not been computed yet. If the nodes are numbered such that $\underline{H}_c^t$ is zero on and above the main diagonal, then the node signal value $y_1(k)$ can be computed as all of the elements h_{m1c} are zero. Consequently, the evaluation of $y_1(k)$ does not require that any of the values in $\underline{y}(k)$ be known. For all other values $y_i(k)$, $2 \leq i \leq N$, the corresponding elements h_{mic} for $m = i,i+1,\ldots,N$ are equal to zero. Consequently, the computation of $y_i(k)$ does not require that any values $y_n(k)$ for $n \geq i$ be known. These values are known, however, as the node values are being computed in consecutive order. Consequently, all values of $y_i(k)$ can be computed consecutively when $\underline{H}_c^t$ is zero on and above the main diagonal. Thus, the proof of Theorem 2.2 is completed. Q.E.D.

Corollary 2.1: An elementary network is computable if and only if the nodes in the network can be numbered in an order such that $\underline{H}_c^t$ is zero on and above the main diagonal. The proof of Corollary 2.1 follows directly from Theorems 2.1 and 2.2.

From Theorems 2.1, 2.2, and Corollary 2.1, it follows that the condition that an elementary network is computable is synonymous with the condition that it contains no closed loops without delay and is also synonymous with the condition that the nodes can be numbered in an order such that, in the matrix representation, $\underline{H}_c^t$ is zero on and above the main diagonal. A matrix representation of an elementary network will be defined as being in computable form if $\underline{H}_c^t$ satisfies the above condition. In general, there may be more than one ordering of the nodes in a computable network which results in a computable form for the matrix representation.

2.6 A MATRIX REPRESENTATION FOR COMPUTABLE NONRECURSIVE ELEMENTARY NETWORK STRUCTURES

An elementary digital network will be defined to be a nonrecursive network if it contains no closed loops that have delay. That is, if we consider all of the regular branches in the network, both coefficient and coefficient and delay branches, the network is said to be nonrecursive if there is no way to choose a directed, connected path through the regular branches which is always in the same direction as the branches or always in the opposite direction of the branches such that it closes upon itself and includes at least one branch with a delay. If at least one such path can be found then the network will be defined as being a recursive network. Furthermore, an elementary network will be defined as a

feedback free network if it is both nonrecursive and computable. That is, a feedback free network contains no closed loops whatsoever, either with or without delay.

Theorem 2.3: If and only if an elementary network is both computable and nonrecursive (that is, if it is feedback free), then the nodes in the network can be numbered in an order such that both matrices $\underline{H}_c^t$ and $\underline{H}_d^t$ in the matrix representation are simultaneously zero on and above their main diagonals.

Proof: The "if" part of the proof can be shown by constructing the network in a form similar to that of Fig. 2.10 with the exception that all regular branches are used in the construction instead of just coefficient branches. Initial nodes in this construction correspond to nodes which have no regular branches, either coefficient or coefficient and delay, entering them. By numbering the nodes in a similar manner as in the first part of the proof of Theorem 2.2, we can construct a matrix representation such that both $\underline{H}_c^t$ and $\underline{H}_d^t$ are simultaneously zero on and above the main diagonal. As the arguments for justifying this procedure are the same as those used in the first parts of the proofs of Theorems 2.1 and 2.2 with regular branches replacing coefficient branches, they will not be restated here.

The "only if" part of the proof can be demonstrated by proving the inverse. That is, if there is a way of numbering the nodes in the network such that $\underline{H}_c^t$ and $\underline{H}_d^t$ are zero on and above their main diagonals, then there are no closed loops

of any kind in the network and, consequently, the network is feedback free. This can be demonstrated by drawing the network in the configuration depicted in Fig. 2.11. The nodes in this configuration are located consecutively from 1 to N on a straight line segment. Because all h_{mnc} and h_{mnd} are zero for $m \geq n$, all of the branches must be directed in the direction of increasing node numbers and no branch can return to the node from which it started. No branches with nonzero coefficient values can exist in this configuration which are directed in the direction of decreasing node numbers. Therefore, if we start at any node in the network and follow any connected, directed path which is always in the same direction as the branches, then it is not possible to return to the node from which we started. That is, we can never find a closed path or a loop in the topology of the network and, consequently, it is feedback free. This completes the proof of Theorem 2.3 Q.E.D.

A matrix representation of an elementary network will be defined to be in <u>feedback free form</u> if $\underline{H}_c^t$ and $\underline{H}_d^t$ are zero on and above the main diagonal. An interesting consequence of this representation is that the matrix sum, $\underline{I}-\underline{H}_c^t-\underline{H}_d^t z^{-1}$, is lower triangular. From matrix theory [11] it can be shown that the inverse of a lower triangular matrix is also lower traingular. Therefore, with the aid of (2.14), it can be observed that the transpose of the transfer function matrix, $\underline{T}^t(z)$, will be lower triangular if the matrix representation

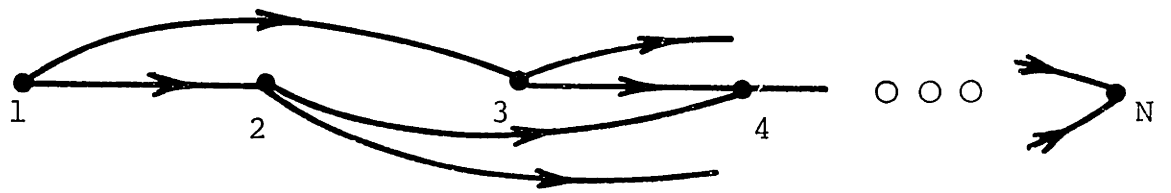


Fig. 2.11 Illustration of a network in feedback free form.

is in feedback free form.

2.7 A STATE SPACE FORMULATION FOR ELEMENTARY NETWORKS

If we consider the node values of an elementary network to be state variables of the network, then it is possible to formulate a set of equations from (2.15) which look similar to a state space representation. This can be accomplished by writing (2.15) in the form,

$$(\underline{I} - \underline{H}_c^t)\underline{y}(k) = \underline{H}_d^t\underline{y}(k - 1) + \underline{u}(k). \tag{2.19}$$

If $\underline{I}-\underline{H}_c^t$ is nonsingular, we can solve for $\underline{y}(k)$ as

$$\underline{y}(k) = (\underline{I} - \underline{H}_c^t)^{-1}\ \underline{H}_d^t\underline{y}(k - 1) + (\underline{I} - \underline{H}_c^t)^{-1}\ \underline{u}(k). \tag{2.20}$$

If we make the definitions

$$\underline{A} = (\underline{I} - \underline{H}_c^t)^{-1}\ \underline{H}_d^t \tag{2.21}$$

and

$$\underline{B} = (\underline{I} - \underline{H}_c^t)^{-1}\ , \tag{2.22}$$

we can then express (2.20) in the form

$$\underline{y}(k) = \underline{A}\ \underline{y}(k - 1) + \underline{B}\ \underline{u}(k). \tag{2.23}$$

This form of the equation looks very similar to a state space representation. Note that with this formulation the number of states (nodes) may be greater than the number of essential states actually necessary to represent the network.

The formulation does, however, allow us to use discrete-time state space techniques [12], [13] in the analysis of elementary digital networks.

If it is assumed that (2.15) is in computable form, then $\underline{I}-\underline{H}_c^t$ is lower triangular and unity along the main diagonal and, therefore, its determinant is unity. That is,

$$\det(\underline{I} - \underline{H}_c^t) = 1. \tag{2.24}$$

Consequently, $\underline{I}-\underline{H}_c^t$ is always nonsingular for networks which are computable, and it is always possible to write the above state space formulation for computable networks.

The "time" domain solution of (2.23) can be determined by standard state space techniques. Consider first the homogeneous or undriven network such that $\underline{u}(k) = \underline{0}$ with initial conditions at time k_o-1 as $\underline{y}(k_o-1)$. Then we can write,

$$\underline{y}(k) = \underline{A}\,\underline{y}(k - 1). \tag{2.25}$$

The state $\underline{y}(k)$ of this system for some k, $k \geq k_o$, can be determined by iterating (2.25) $k-k_o+1$ times. This gives,

$$\underline{y}(k) = \underline{\Phi}(k-k_o+1)\ \underline{y}(k_o-1), \tag{2.26}$$

where $\underline{\Phi}(r)$ is defined as

$$\underline{\Phi}(r) = \underline{A}^r. \tag{2.27}$$

The matrix $\underline{\Phi}(r)$ is often referred to in state space terminology [14] as the <u>state transition matrix</u>, where

$$\underline{\Phi}(0) = \underline{A}^{o} = \underline{I}. \tag{2.28}$$

If the network is driven with a set of sources $\underline{u}(k)$, then (2.26) will take the form

$$\underline{y}(k) = \underline{\Phi}(k-k_o)\ \underline{y}(k_o) + \sum_{\ell=k_o}^{k} \underline{\Phi}(k-\ell)\ \underline{B}\ \underline{x}_s(\ell) \tag{2.29}$$

where $k \geq k_o$. This equation is often referred to as the variation of constants formula [14]. The first term on the right side of the equation can be identified as the homogeneous response to the initial conditions and the second term can be identified as the particular response to the driving sources $\underline{u}(k)$. The particular response is simply the convolution sum written in matrix form where

$$\underline{t}^{t}(k) = \underline{\Phi}(k)\ \underline{B} \tag{2.30}$$

can be defined as the impulse response matrix of the network.

The complex frequency domain solution for the state space representation can be derived by taking the z transform of (2.23). This can be written as

$$\underline{Y}(z) = \underline{A}\ \underline{Y}(z)\ z^{-1} + \underline{B}\ \underline{U}(z). \tag{2.31}$$

By collecting terms and premultiplying by $z(\underline{I}z-\underline{A})^{-1}$, we get

$$\underline{Y}(z) = z(\underline{I}z - \underline{A})^{-1}\ \underline{B}\ \underline{U}(z). \tag{2.32}$$

Equation (2.32) can be identified as the z transform of (2.29)

for the case where $k_o \to \infty$ and assuming that the system is stable such that the homogeneous response has decayed to zero. By comparing (2.32) with (2.8), we can identify the term $z(\underline{I}z-\underline{A})^{-1}\underline{B}$ as the matrix of transfer functions in the network. That is,

$$\underline{T}^t(z) = z(\underline{I}z - \underline{A})^{-1}\underline{B}. \tag{2.33}$$

From linear algebra [11] or state space theory [12], [13], [14], it can be shown that the problem of determining the poles of the transfer functions reduces to that of determining the eigenvalues of the matrix $\underline{A}$. Furthermore, it can be shown that these eignevalues are roots of the characteristic polynomial defined by

$$\det(\underline{I}z - \underline{A}) = 0. \tag{2.34}$$

This polynomial is the common denominator of all of the transfer functions in $\underline{T}(z)$. The poles of the individual transfer functions are the factors of this polynomial which do not cancel with their respective numerator factors.

2.8 NONELEMENTARY NETWORK REPRESENTATIONS, EQUIVALENT NETWORKS, AND THEIR USES

While elementary network representations are useful for representing, explicitly, all internal signals and operations in a network structure, they are not always the most useful representations for illustrating overall topological

properties of a network such as modularity or the general structural form of a network. In such cases, a nonelementary network representation may prove to be a simpler and more manageable way of looking at the structure. In general, branch transmittances of nonelementary networks may correspond to more complicated functions than gains or gains and delays. It will be assumed, however, that they will still correspond to causal linear shift invariant operations.

A set of concepts which lead to the formulation of nonelementary network representations is that of equivalent networks. As in conventional circuit theory, where complicated networks can be represented by their Norton or Thevenin equivalent circuits, so can complicated flow graphs be represented by equivalent networks. Also, as in conventional circuit theory, once a digital network is replaced by its equivalent network representation, all information about its internal variables and structure is forfeited in the representation.

The concept of representing a single input, single output subnetwork within a larger network by its equivalent network representation is illustrated in Fig. 2.12. The subnetwork and its connecting branches can be replaced by an equivalent network which consists of a regular branch and a source branch. The branch transmittance function C(z) of the regular branch is taken as the system function $T_{pq}(z)$ of the isolated subnetwork (disconnected from the main network).

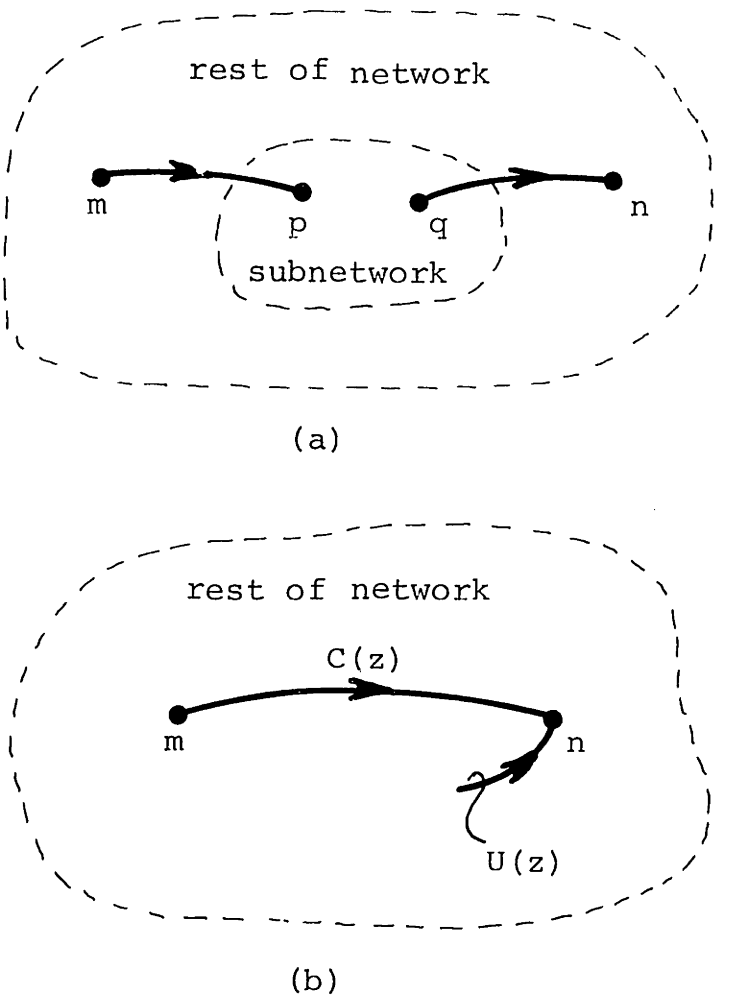


Fig. 2.12 (a) A network with a subnetwork.
(b) The subnetwork replaced by its equivalent network representation.

That is,

$$C(z) = T_{pq}(z)\Big|_{\substack{\text{isolated}\\ \text{subnetwork}}} . \tag{2.35}$$

The source branch value U(z) of the equivalent network represents the contribution of sources internal to the subnetwork and is given as

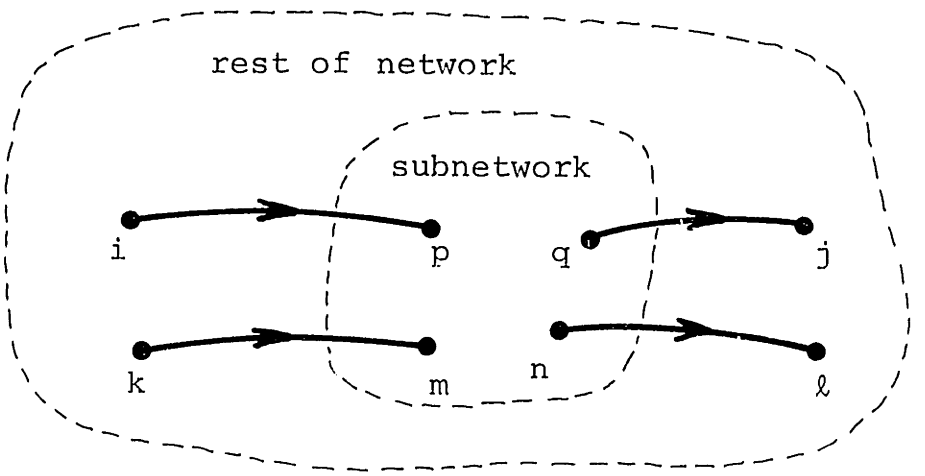
$$U(z) = \sum_{\substack{i\\ \text{Sum over the}\\ \text{isolated subnetwork}}} T_{ip}(z)\ U_i(z). \tag{2.36}$$

The concept of equivalent networks can be extended to subnetworks with any number of specified inputs and outputs. A particularly interesting case is that of subnetworks with two specified inputs and two specified outputs. The use of an equivalent network to replace a subnetwork and its connecting branches for this case is depicted in Fig. 2.13.

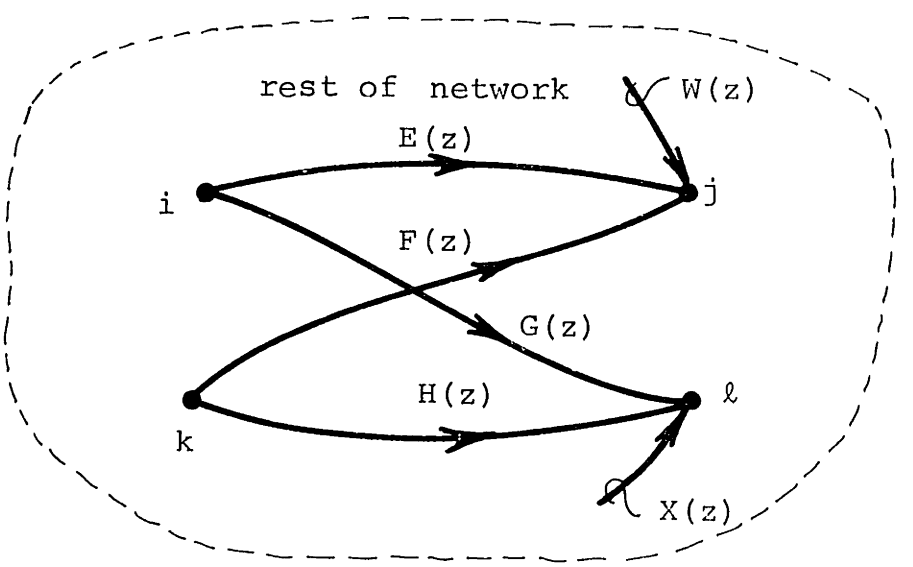
The parameters of the equivalent network in Fig. 2.13 can be determined in a manner similar to that for the single input, single output case. They are given as

$$E(z) = T_{pq}(z)\Big|_{\substack{\text{isolated}\\ \text{subnetwork}}} , \tag{2.37}$$

$$F(z) = T_{mq}(z)\Big|_{\substack{\text{isolated}\\ \text{subnetwork}}} , \tag{2.38}$$



(a)



(b)

Fig. 2.13 (a) A network with a two input, two output subnetwork.

(b) The equivalent network representation of the subnetwork.

$$G(z) = T_{pn}(z)\Big|_{\substack{\text{isolated}\\ \text{subnetwork}}}, \tag{2.39}$$

$$H(z) = T_{mn}(z)\Big|_{\substack{\text{isolated}\\ \text{subnetwork}}}, \tag{2.40}$$

$$W(z) = \sum_{\substack{i\\ \text{isolated}\\ \text{subnetwork}}} T_{iq}(z)\ U_i(z), \tag{2.41}$$

and

$$X(z) = \sum_{\substack{i\\ \text{isolated}\\ \text{subnetwork}}} T_{in}(z)\ U_i(z). \tag{2.42}$$

A special subclass of the two-input two-output equivalent networks is of particular importance in the structural concepts of the digital ladder networks of Fettweis [15], [16], [17] and also, the ladder networks of Mitra and Sherwood [18]. For this subclass, all of the internal sources of the subnetwork are assumed to be zero and the inputs and outputs are taken as input output pairs. This class of equivalent networks is depicted in Fig. 2.14 where the nodes i and j are taken as one input output pair and the nodes k and ℓ are taken as the second input output pair.

The relationship between the branch signals $X_\ell(z)$ $X_j(z)$ and the node signals $Y_i(z)$ and $Y_k(z)$ in Fig. 2.14 can be

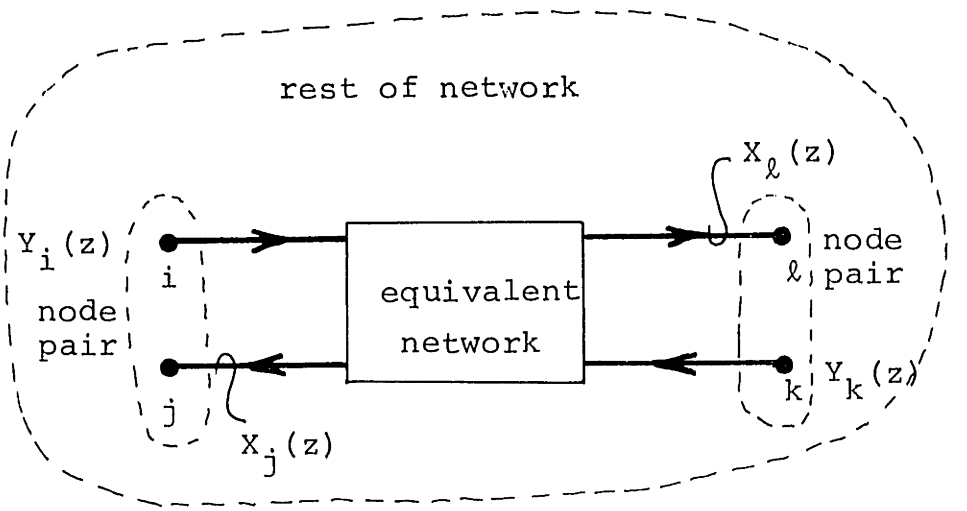


Fig. 2.14 A subclass of two-input two-output equivalent networks

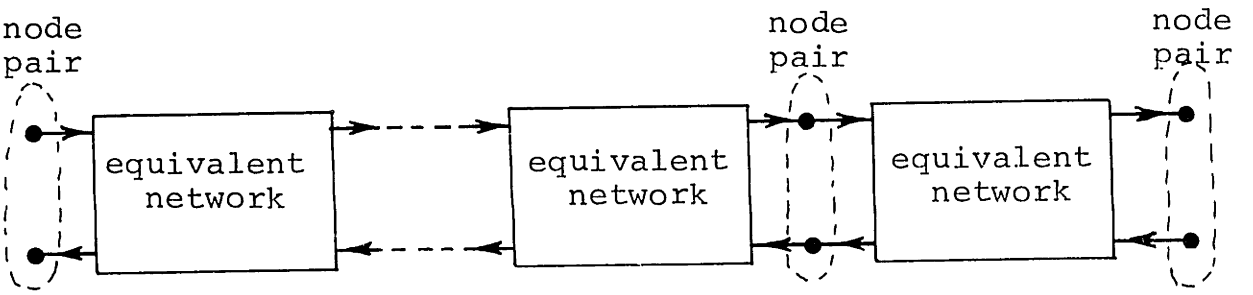


Fig. 2.15 Representation of a ladder network in terms of cascaded equivalent networks of the form in Fig. 2.14.

expressed with the aid of relations (2.37) to (2.40) as

$$\begin{bmatrix} X_j(z) \\ X_\ell(z) \end{bmatrix} = \begin{bmatrix} E(z) & F(z) \\ G(z) & H(z) \end{bmatrix} \begin{bmatrix} Y_i(z) \\ Y_k(z) \end{bmatrix} . \tag{2.43}$$

This type of substructure serves as the basic building block in the ladder networks where equivalent networks are connected or cascaded by node pairs instead of single nodes as depicted in Fig. 2.15.

This class of structures, as depicted in Fig. 2.15, very closely parallels the concepts of two-port network theory in conventional circuit theory. This type of structure in conventional circuit theory serves as one of the primary classes of structures used for classical filter synthesis. The relation (2.43) is closely related to the two-port scattering matrix in conventional circuit theory.

Another useful way to characterize a two-port network in conventional circuit theory is by the chain matrix. This concept can be applied to the class of filters in Fig. 2.15 by rewriting (2.43) for a single subnetwork as

$$\begin{bmatrix} Y_i(z) \\ X_j(z) \end{bmatrix} = \begin{bmatrix} \frac{1}{G(z)} & -\frac{H(z)}{G(z)} \\ \frac{E(z)}{G(z)} & \frac{F(z)G(z) - E(z)H(z)}{G(z)} \end{bmatrix} \begin{bmatrix} X_\ell(z) \\ Y_k(z) \end{bmatrix} \tag{2.44}$$

where it is assumed that G(z) does not equal zero. A very useful property of this chain matrix representation of the blocks in Fig. 2.15 is that the chain matrix of the overall

network is simply the matrix product of the chain matrices of the individual blocks taken in the order in which they appear in the network.

The concept of an equivalent network can be applied in general to the networks or subnetworks with any number of inputs and outputs. This concept is, in a sense, a generalization of the concept of a scattering matrix representation of a network or subnetwork for signal-flow graphs.

Chapter III

SOME BASIC SENSITIVITY RELATED THEOREMS AND PROPERTIES OF DIGITAL NETWORKS

3.1 INTRODUCTION

In this chapter a number of basic theorems and properties relating to digital networks are presented. Specifically, those theorems and properties concerned with coefficient sensitivities will be considered. Many of these theorems have similar analogs in conventional circuit theory. Also many of them, although specifically related to digital networks in this discussion, apply to signal-flow graphs in general.

3.2 TELLEGEN'S THEOREM FOR DIGITAL NETWORKS

For digital networks and signal-flow graphs there exist a number of theorems which closely resemble Tellegen's theorem in conventional circuit theory [19]. Several formulations and variations of this theorem for signal-flow graphs of digital networks have been recently proposed by a number of authors including Seviora and Sablatash [20], Fettweis [21], and Lee [22], [23]. The form of Tellegen's theorem which will be used here is developed to conform with the network terminology defined in Chapter II and is perhaps most similar to the form of the theorem proposed by Fettweis. The

theorem in this form closely resembles the difference form of Tellegen's theorem in conventional circuit theory.

Tellegen's Theorem for Digital Networks: Consider two networks with N nodes each, labeled from 1 through N, which are not necessarily required to have the same topology. For the first network, using the notation of Chapter 2, we will denote the node signals, total branch signals, and source branch signals as $Y_n(z)$, $X_{mn}(z)$, and $U_n(z)$, respectively, and for the second network as $Y'_n(z)$, $X'_{mn}(z)$, and $U'_n(z)$, respectively. Then Tellegen's theorem can be stated as

$$\sum_{n=1}^{N} \sum_{m=1}^{N} \left(Y_n(z)\ X'_{mn}(z) - Y'_n(z)\ X_{mn}(z) \right) + \sum_{n=1}^{N} \left(Y_n(z)\ U'_n(z) - Y'_n(z)\ U_n(z) \right) = 0. \qquad (3.1)$$

Proof: From Chapter 2, we can write

$$Y_n(z) = U_n(z) + \sum_{m=1}^{N} X_{mn}(z) \qquad (3.2a)$$

for the unprimed system and

$$Y'_n(z) = U'_n(z) + \sum_{m=1}^{N} X'_{mn}(z) \qquad (3.2b)$$

for the primed system. We can them rewrite relation (3.1) in the form

$$\sum_{n=1}^{N} \left[Y_n(z) \sum_{m=1}^{N} \left(X'_{mn}(z) \right) - Y'_n(z) \sum_{m=1}^{N} \left(X_{mn}(z) \right) + Y_n(z)\, U'_n(z) - Y'_n(z)\, U_n(z) \right] = 0$$

or

$$\sum_{n=1}^{N} \left[Y_n(z) \left\{ U'_n(z) + \sum_{m=1}^{N} \left(X'_{mn}(z) \right) \right\} - Y'_n(z) \left\{ U_n(z) + \sum_{m=1}^{N} \left(X_{mn}(z) \right) \right\} \right] = 0.$$

By applying relations (3.2a) and (3.2b), we get

$$\sum_{n=1}^{N} \left[Y_n(z)\, Y'_n(z) - Y'_n(z)\, Y_n(z) \right] = 0.$$

This equation is obviously true and the proof is completed.

Q.E.D.

Tellegen's theorem can also be written in the form of (3.1) for the signal variables expressed in the discrete time domain instead of their z transforms. The proof is similar to that above.

A number of observations concerning Tellegen's theorem can be made. The two sets of variables, the primed set and the unprimed set, can represent two different signal distributions in two different networks, in which case the only

requirement on the two networks is that they contain the same number of nodes N. Alternatively, the two signal distributions can correspond to two different signals on the same network at different times. The left term in (3.1) corresponds to a summation over all regular branches, and the right term corresponds to a summation over all source branches. In the derivation of Tellegen's theorem it is not necessary to assume that the branch operations are linear or shift invariant. Consequently, Tellegen's theorem is also applicable to nonlinear and shift variant networks as well.

A second form of Tellegen's theorem which will be of interest to us is the form

$$\sum_{n=1}^{N} \sum_{m=1}^{N} \left[Y_n(z)\, X'_{mn}(z) - Y'_m(z)\, X_{nm}(z) \right] + \sum_{n=1}^{N} Y_n(z)\, U'_n(z) - \sum_{m=1}^{N} Y'_m(z)\, U_m(z) = 0. \tag{3.3}$$

The use and restrictions of this equation are the same as those for (3.1). The proof goes as follows:

Rewriting (3.3) in the form

$$\sum_{n=1}^{N} \left[Y_n(z) \left\{ \sum_{m=1}^{N} X'_{mn}(z) + U'_n(z) \right\} \right] - \sum_{m=1}^{N} \left[Y_m(z) \left\{ \sum_{n=1}^{N} \left(X_{nm}(z) \right) + U_m(z) \right\} \right] = 0$$

and applying relation (3.2), we get

$$\sum_{n=1}^{N} \left[Y_n(z)\ Y'_n(z) \right] - \sum_{m=1}^{N} \left[Y'_m(z)\ Y_m(z) \right] = 0.$$

The subscript m in the right term can be changed to n and again the relation is obviously true. Q.E.D.

As in the first form of Tellegen's theorem, this form can also be written for the signal variables in the discrete time domain.

3.3 THE CONCEPTS OF RECIPROCITY AND INTERRECIPROCITY

A digital network is defined to be reciprocal [21] if for any two signal distributions (primed and unprimed sets) we can write

$$\sum_{n=1}^{N} \left[Y_n(z)\ U'_n(z) - Y'_n(z)\ U_n(z) \right] = 0 \qquad (3.4)$$

In words the concept of reciprocity can be stated as follows: If we excite a homogeneous network with a single source U(z) at a node m and observe an output Y(z) at a node n, then move the source to node n and observe the same output Y(z) at node m, the network is reciprocal between nodes m and n. For a linear shift-invariant network this implies that $T_{mn}(z) = T_{nm}(z)$. If relation (3.4) is satisfied, then the network is reciprocal between all of its nodes. This can be shown by applying the relation

$$Y_n(z) = \sum_{m=1}^{N} T_{mn}(z)\ U_m(z) \tag{3.5}$$

to the definition of reciprocity which gives

$$\sum_{n=1}^{N} \left[U'_n(z) \sum_{m=1}^{N} \left(T_{mn}(z)\ U_m(z) \right) \right] -$$

$$\sum_{n=1}^{N} \left[U_n(z) \sum_{m=1}^{N} \left(T'_{mn}(z)\ U'_m(z) \right) \right] = 0.$$

By interchanging the subscripts m and n in the second term and combining terms, this expression can be stated as

$$\sum_{n=1}^{N} \sum_{m=1}^{N} U'_n(z)\ U_m(z) \left(T_{mn}(z) - T'_{nm}(z) \right) = 0. \tag{3.6}$$

Because the same network is used for the primed and unprimed varaibles, $T'_{nm}(z)$ must be the same as $T_{nm}(z)$. Relation (3.6) must be true for any two signal variables $U'_n(z)$ and $U_m(z)$ and, therefore, for a reciprocal network

$$T_{mn}(z) = T_{nm}(z) \tag{3.7}$$

This implies that the transfer function matrix $\underline{T}(z)$ must be symmetric.

The condition of reciprocity is generally not found in practical digital signal processing networks, especially

reciprocity with respect to all of the nodes. A closely related but more important concept for digital networks is that of interreciprocity. Two networks are defined to be interreciprocal [21] if for any two signal distributions, one on each network, relation (3.4) is satisfied. In this case, the primed and the unprimed sets of variables correspond to the primed and unprimed networks. It is important to recognize the distinction that for reciprocity the primed and unprimed variables in (3.4) correspond to two different signal distributions in the same network at different times, whereas for interreciprocity, the primed and unprimed variables in (3.4) correspond to two different signal distributions in two different networks. For interreciprocity we can again apply relation (3.5) to (3.4) to arrive at (3.6). From equation (3.6) it can be established that if two networks are interreciprocal, then their transfer functions satisfy the relation

$$T_{mn}(z) = T'_{nm}(z). \tag{3.8}$$

This implies that the transfer function matrices $\underline{T}(z)$ and $\underline{T}'(z)$ are the transpose of each other. The concept of interreciprocity will be a useful analysis tool in later sections.

Another relationship concerning reciprocity or interreciprocity can be observed by substituting definition (3.4) into Tellegen's theorem (3.1). This implies the relationship

$$\sum_{n=1}^{N} \sum_{m=1}^{N} \left(Y_n(z)\ X'_{mn}(z) - Y'_n(z)\ X_{mn}(z) \right) = 0. \qquad (3.9)$$

In other words, the condition of reciprocity or interreciprocity implies that the two terms in Tellegen's theorem, the sum over the source branches and the sum over the regular branches, must both be zero independently.

3.4 THE CONCEPT OF TRANSPOSITION OF A NETWORK

The transpose of a digital network or a signal-flow graph is defined as the operation of reversing the direction of all the regular branches in the network [21], [24]. This has the same effect as taking the matrix transpose of the total branch transmittance matrix $\underline{H}(z)$. That is,

$$H'_{mn}(z) = H_{nm}(z) \qquad (3.10)$$

where the primed network is assigned to be the transpose network.

A theorem can now be stated that a network and its transpose are interreciprocal [21]. The theorem can be proved with the aid of the relation (see Chapter 2)

$$Y_n(z) = U_n(z) + \sum_{m=1}^{N} H_{mn}(z)\ Y_m(z) \qquad (3.11)$$

and the definition of interreciprocity (3.4). Equation (3.11) can be rewritten in the form

$$U_n(z) = Y_n(z) - \sum_{m=1}^{N} H_{mn}(z)\ Y_m(z).$$

Upon substituting into (3.4) for $U_n(z)$ and $U_n'(z)$, we get

$$\sum_{n=1}^{N} \left[Y_n(z) \left\{ Y_n'(z) - \sum_{m=1}^{N} \left(H_{mn}'(z)\ Y_m'(z) \right) \right\} - \right.$$

$$\left. Y_n'(z) \left\{ Y_n(z) - \sum_{m=1}^{N} H_{mn}(z)\ Y_m(z) \right\} \right] = 0.$$

Cancelling terms and rearranging gives

$$\sum_{n=1}^{N} \sum_{m=1}^{N} \left[- Y_n(z)\ Y_m'(z)\ H_{mn}'(z) \right] +$$

$$\sum_{n=1}^{N} \sum_{m=1}^{N} \left[Y_n'(z)\ Y_m(z)\ H_{mn}(z) \right] = 0.$$

Finally, interchanging subscripts m and n in the second term and combining terms gives

$$\sum_{n=1}^{N} \sum_{m=1}^{N} \left[Y_n(z)\ Y_m'(z) \left(H_{mn}'(z) - H_{nm}(z) \right) \right] = 0. \qquad (3.12)$$

From this expression it can be seen that if the primed and unprimed networks are the transpose of each other as

defined by (3.10), then the condition of interreciprocity will be satisfied and the proof is completed.

3.5 A FIRST-ORDER NETWORK SENSITIVITY RELATION

A set of parameters that we would often like to know in the implementation of a structure is the sensitivity of the system function from input to output to the various branch coefficients or branch transmittance functions. Using Tellegen's theorem and the concepts of interreciprocity and transposition, a very convenient expression for evaluating these coefficient sensitivities can be derived. The derivation which will be given here follows closely the procedure proposed by Fettweis [21]. The same expression, however, can also be derived using other forms of Tellegen's theorem and the concepts of adjoint networks [20], [22], [23].

Consider a network with a single input specified to be at node k and a single output specified to be the node signal value of node ℓ. Then, the system function of interest is $T_{k\ell}(z)$ and we want to determine the first-order sensitivity of this system function with respect to some branch transmittance function $H_{pq}(z)$ of a branch directed from some node p to some node q (see Figure 3.1a). That is, we want to determine

$$\left.\begin{array}{l}\text{Sensitivity of } T_{k\ell}(z) \\ \text{with respect to } H_{pq}(z)\end{array}\right\} = \frac{\partial\, T_{k\ell}(z)}{\partial\, H_{pq}(z)} \,. \tag{3.14}$$

To do this, consider the three networks of Fig. 3.1.

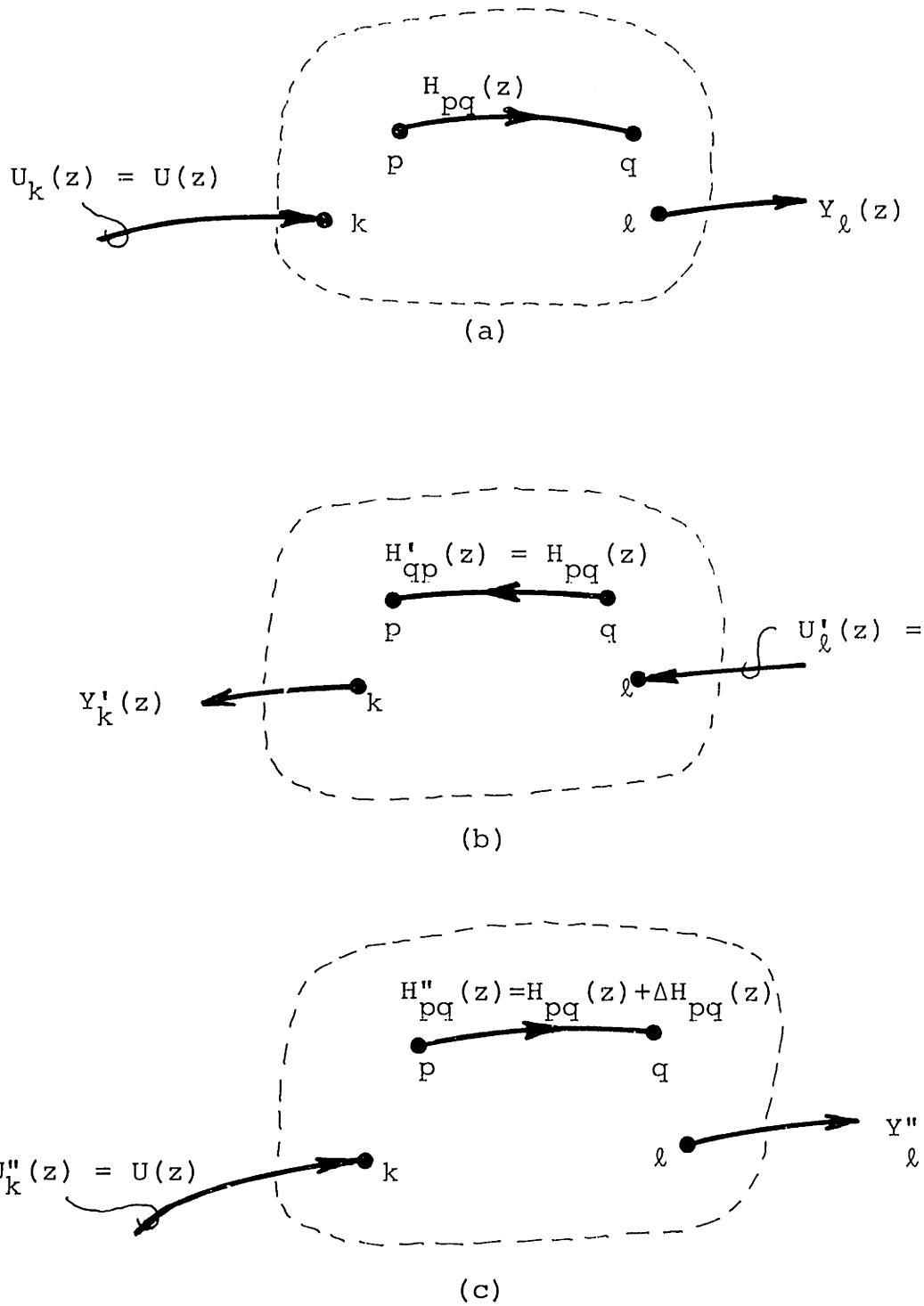


Fig. 3.1 Networks required for the derivation of the first-order network sensitivities, (a) original network (unprimed system), (b) transpose of the original network (primed system), and (c) modified original network (double primed system).

The first network is the original network and it is specified by unprimed variables. The second network is the transpose of the original network and it is specified by primed variables. The third network is the original network with the exception that the branch transmittance $H_{pq}(z)$ is changed by a small amount $\Delta H_{pq}(z)$. This network is specified by double primed variables. Each network is driven by a single identical source $U(z)$ as depicted in Fig. 3.1.

We begin with the form of Tellegen's theorem given by (3.3). Written for the primed and double primed systems, this relation takes the form

$$\sum_{n=1}^{N} \sum_{m=1}^{N} \left[Y'_n(z)\ X''_{mn}(z) - Y''_m(z)\ X'_{nm}(z) \right] + \sum_{n=1}^{N} \left[Y'_n(z)\ U''_n(z) \right] - \sum_{m=1}^{N} \left[Y''_m(z)\ U'_m(z) \right] = 0. \tag{3.15}$$

From Chapter 2, we can show that

$$X''_{mn}(z) = H''_{mn}(z)\ Y''_m(z) \tag{3.16}$$

and

$$X'_{nm}(z) = H'_{nm}(z)\ Y'_n(z). \tag{3.17}$$

Applying these two relations to (3.15) and collecting terms gives

$$\sum_{n=1}^{N}\sum_{m=1}^{N}\left[Y'_n(z)\ Y''_m(z)\ \left\{H''_{mn}(z) - H'_{nm}(z)\right\}\right] + \sum_{n=1}^{N}\left[Y'_n(z)\ U''_n(z)\right]$$

$$- \sum_{m=1}^{N}\left[Y''_m(z)\ U'_m(z)\right] = 0. \qquad (3.18)$$

From Fig. 3.1, it can be determined that the primed system and the double primed system are transpose networks with the exception of one branch $H''_{pq}(z)$. Therefore, for all of the branches, with the exception of $H''_{pq}(z)$, it is known from the definition of transpose networks that

$$H''_{mn}(z) - H'_{nm}(z) = 0,$$

and for the branch $H''_{pq}(z)$, it can be seen that

$$H''_{pq}(z) - H'_{qp}(z) = H_{pq}(z) + \Delta H_{pq}(z) - H'_{qp}(z) = \Delta H_{pq}(z).$$

Therefore, all of the summations in the right term of (3.18) add to zero with one exception. The second and third summations in (3.18) each give a single term because only a single source is allowed in each network. Therefore, the evaluation of (3.18) gives only the three terms

$$Y'_q(z)\ Y''_p(z)\ \Delta H_{pq}(z) + Y'_k(z)\ U(z) - Y''_\ell(z)\ U(z) = 0. \qquad (3.19)$$

From (3.5) and Fig. 3.1, the following relations can be

determined:

$$Y'_k(z) = T'_{\ell k}(z)\ U(z) = T_{k\ell}(z)\ U(z),$$

$$Y'_\ell(z) = T''_{k\ell}(z)\ U(z) = \left[T_{k\ell}(z) + \Delta T_{k\ell}(z)\right]\ U(z),$$

$$Y'_q(z) = T'_{\ell q}(z)\ U(z) = T_{q\ell}(z)\ U(z),$$

and

$$Y''_p(z) = T''_{kp}(z)\ U(z).$$

Applying these relations to (3.19) and cancelling $(U(z))^2$ gives

$$T_{q\ell}(z)\ T''_{kp}(z)\ \Delta H_{pq}(z) + T_{k\ell}(z) - \left[T_{k\ell}(z) + \Delta T_{k\ell}(z)\right] = 0. \tag{3.20}$$

By cancelling terms and rearranging, we get

$$\frac{\Delta T_{k\ell}(z)}{\Delta H_{pq}(z)} = T_{q\ell}(z)\ T''_{kp}(z) \tag{3.21}$$

Finally, taking the limit as $\Delta H_{pq}(z)$ goes to zero, the desired relation for (3.14) can be determined as

$$\lim_{\Delta H_{pq}(z)\to 0} \frac{\Delta T_{k\ell}(z)}{\Delta H_{pq}(z)} = \left[\lim_{\Delta H_{pq}(z)\to 0} T''_{kp}(z)\right] T_{q\ell}(z)$$

or

$$\frac{\partial T_{k\ell}(z)}{\partial H_{pq}(z)} = T_{kp}(z)\ T_{q\ell}(z). \tag{3.22}$$

The desirable feature of expression (3.22) is that it relates the sensitivity of the system function, with respect to the branch transmittance functions, in terms of other transfer functions in the structure.

For the case of elementary networks, the branches are restricted to be coefficients or coefficients and delays. Relation (3.22) can be used to express the sensitivity of the system function to these coefficients. For a branch from node p to node q composed of a single coefficient c, (3.22) can be applied directly to obtain the coefficient sensitivity,

$$\frac{\partial T_{k\ell}(z)}{\partial c} = T_{kp}(z)\ T_{q\ell}(z). \tag{3.23}$$

For a branch from node p to node q composed of a coefficient d and a delay, the chain rule must be used in conjunction with (3.22) to determine the coefficient sensitivity. That is,

$$H_{pq}(z) = dz^{-1}$$

and

$$\frac{\partial T_{k\ell}(z)}{\partial d} = \frac{\partial T_{k\ell}(z)}{\partial H_{pq}(z)} \cdot \frac{\partial H_{pq}(z)}{\partial d} = T_{kp}(z)\ T_{q\ell}(z)\ z^{-1}. \tag{3.24}$$

3.6 HIGHER-ORDER NETWORK SENSITIVITIES AND RELATIONS

In the previous section, a fundamental relation was derived which expresses the sensitivity of a transfer function with respect to a branch transmittance function in terms of two other transfer functions in the network. In this section, this relation together with the chain rule for partial differentiation will be used to derive higher-order sensitivity expressions.

For example, the second-order sensitivity can be obtained by taking the partial derivative of the first-order sensitivity which gives

$$\frac{\partial^2 T_{k\ell}(z)}{\partial H^2_{pq}(z)} = \frac{\partial}{\partial H_{pq}(z)} \left[\frac{\partial T_{k\ell}(z)}{\partial H_{pq}(z)}\right]$$

$$= \frac{\partial}{\partial H_{pq}(z)} [T_{kp}(z)\ T_{q\ell}(z)]. \qquad (3.25)$$

Applying the chain rule gives

$$\frac{\partial^2 T_{k\ell}(z)}{\partial H^2_{pq}(z)} = T_{kp}(z)\ \frac{\partial T_{q\ell}(z)}{\partial H_{pq}(z)} + T_{q\ell}(z)\ \frac{\partial T_{kp}(z)}{\partial H_{pq}(z)}\ .$$

By applying the first-order sensitivity relation (3.22) to each of the partial derivatives on the left side of the equation, we get

$$\frac{\partial^2 T_{k\ell}(z)}{\partial H^2_{pq}(z)} = T_{kp}(z)T_{qp}(z)T_{q\ell}(z) + T_{q\ell}(z)T_{kp}(z)T_{qp}(z).$$

Then by combining terms, the second-order sensitivities can be expressed in the form

$$\frac{\partial^2 T_{k\ell}(z)}{\partial H^2_{pq}(z)} = 2T_{qp}(z)\ T_{kp}(z)\ T_{q\ell}(z). \tag{3.26}$$

An alternate form for this expression can be determined by again applying (3.22) to give

$$\frac{\partial^2 T_{k\ell}(z)}{\partial H^2_{pq}(z)} = 2\,T_{qp}(z)\ \frac{\partial T_{k\ell}(z)}{\partial H_{pq}(z)}. \tag{3.27}$$

We can apply a similar procedure to that shown above to compute all higher-order sensitivities. In general, for the n^{th}-order sensitivity, the sensitivity expression can be written in the form

$$\frac{\partial^n T_{k\ell}(z)}{\partial H^n_{pq}(z)} = n!\ T^{n-1}_{qp}(z)\ T_{kp}(z)\ T_{q\ell}(z) \tag{3.28}$$

for $n \geq 1$. This expression can be verified for $n = 1$ and $n = 2$ from relations (3.22) and (3.28), respectively. For an arbitrary n, $n \geq 1$, we can verify the above expression by showing that if it is true for some value n, then it is true

for n + 1. To do this, we can write

$$\frac{\partial^{n+1} T_{k\ell}(z)}{\partial H^{n+1}_{pq}(z)} = \frac{\partial}{\partial H_{pq}} \left[\frac{\partial^{n} T_{k\ell}(z)}{\partial H^{n}_{pq}(z)} \right]. \tag{3.29}$$

By substituting (3.28) into this equation and taking the appropriate derivatives, we get

$$\begin{aligned} \frac{\partial^{n+1} T_{k\ell}(z)}{\partial H^{n+1}_{pq}(z)} &= n!\ T^{n-1}_{qp}(z)\ T_{kp}(z)\ \frac{\partial T_{q\ell}(z)}{\partial H_{pq}(z)} \\ &\quad + n!\ T^{n-1}_{qp}(z)\ T_{q\ell}(z)\ \frac{\partial T_{kp}(z)}{\partial H_{pq}(z)} \\ &\quad + n!\ T_{kp}(z)\ T_{q\ell}(z)(n-1)\ T^{n-2}_{qp}(z)\ \frac{\partial T_{qp}(z)}{\partial H_{pq}(z)}\ . \end{aligned} \tag{3.30}$$

Using the first-order sensitivity expression (3.22), we can write the first-order sensitivities in (3.30) as

$$\frac{\partial T_{q\ell}(z)}{\partial H_{pq}(z)} = T_{qp}(z)\ T_{q\ell}(z),$$

$$\frac{\partial T_{kp}(z)}{\partial H_{pq}(z)} = T_{kp}(z)\ T_{qp}(z),$$

and

$$\frac{\partial T_{qp}(z)}{\partial H_{pq}(z)} = T^{2}_{qp}(z).$$

Applying these relations to (3.30) and collecting terms then gives

$$\frac{\partial^{n+1} T_{k\ell}(z)}{\partial H^{n+1}_{pq}(z)} = n!\ T^{n}_{qp}(z)\ T_{kp}(z)\ T_{q\ell}(z)\ (1+1+(n-1))$$

$$= (n+1)!\ T^{n}_{qp}(z)\ T_{kp}(z)\ T_{q\ell}(z). \qquad (3.31)$$

If we make the substitution of variables, $n = n + 1$ (3.31) will be identical in form to (3.28). Consequently, since we have verified (3.28) for $n = 1$ (and $n = 2$) and have demonstrated that if it is true for n, it is also true for $n + 1$, the relation is true for all n such that $n \geq 1$.

We can also imply a recursive relationship between the higher-order sensitivities of a network. By taking the ratio of the $n + 1^{th}$-order sensitivity to the n^{th}-order sensitivity, we get

$$\frac{\dfrac{\partial^{n+1} T_{k\ell}(z)}{\partial H^{n+1}_{pq}(z)}}{\dfrac{\partial^{n} T_{k\ell}(z)}{\partial H^{n}_{pq}(z)}} = \frac{(n+1)!\ T^{n}_{qp}(z)\ T_{kp}(z)\ T_{q\ell}(z)}{n!\ T^{n-1}_{qp}(z)\ T_{kp}(z)\ T_{q\ell}(z)}.$$

By cancelling terms and multiplying by the n^{th}-order sensitivity, we get the recursive relation

$$\frac{\partial^{n+1} T_{k\ell}(z)}{\partial H^{n+1}_{pq}(z)} = (n+1)\ T_{qp}(z)\ \frac{\partial^{n} T_{k\ell}(z)}{\partial H^{n}_{pq}(z)} \qquad (3.32)$$

for $n \geq 1$.

3.7 A LARGE-CHANGE NETWORK SENSITIVITY RELATION

A general expression for all of the higher-order derivatives of a system function with respect to a particular branch transmittance function has been derived. This expression, together with a Taylor's series expansion, can be used to derive a general relation that expresses a system function in terms of the branch transmittance function in question and three transfer functions in the structure. To derive this relation, we begin by writing the Taylor's series expansion of the system function $T_{k\ell}(z)$ about some nominal value of the branch transmittance function $H_{pq}(z)$ denoted by a prime $H'_{pq}(z)$. That is,

$$T_{k\ell}(z)\Big|_{H_{pq}(z)=H'_{pq}(z)+\Delta H_{pq}(z)} = T_{k\ell}(z)\Big|_{H_{pq}(z)=H'_{pq}(z)} + \sum_{n=1}^{\infty} \frac{1}{n!} \left\{ \frac{\partial^n T_{k\ell}(z)}{\partial H^n_{pq}(z)} \right\} \Bigg|_{H_{pq}(z)=H'_{pq}(z)} [\Delta H_{pq}(z)]^n, \quad (3.33)$$

where $\Delta H_{pq}(z)$ represents the deviation of the branch transmittance function $H_{pq}(z)$ from its nominal value $H'_{pq}(z)$.

The general expression for the higher-order derivatives (3.28) can now be applied to (3.33) to give

$$\left.T_{k\ell}(z)\right|_{H_{pq}(z)=H'_{pq}(z)+\Delta H_{pq}(z)} = \left.T_{k\ell}(z)\right|_{H_{pq}(z)=H'_{pq}(z)} + \sum_{n=1}^{\infty} \frac{1}{n!}\left[n!\, T_{qp}^{n-1}(z)T_{kp}(z)T_{q\ell}(z)\right][\Delta H_{pq}(z)]^n, \tag{3.34}$$

where the transfer functions $T_{qp}(z)$, $T_{kp}(z)$, and $T_{q\ell}(z)$ must be evaluated for the branch transmittance function at its nominal value $H'_{pq}(z)$. By cancelling the n! and making the change of variables m = n - 1, (3.34) can be written as

$$\left.T_{k\ell}(z)\right|_{H_{pq}(z)=H'_{pq}(z)+\Delta H_{pq}(z)} = \left.T_{k\ell}(z)\right|_{H_{pq}(z)=H'_{pq}(z)} + T_{kp}(z)T_{q\ell}(z)\Delta H_{pq}(z)\sum_{m=0}^{\infty}[T_{qp}(z)\Delta H_{pq}(z)]^m. \tag{3.35}$$

If the summation in (3.35) converges, then it can be written in closed form as

$$T_{k\ell}(z)\Big|_{H_{pq}(z)=H'_{pq}(z)+\Delta H_{pq}(z)} = T_{k\ell}(z)\Big|_{H_{pq}(z)=H'_{pq}(z)} + \frac{T_{kp}(z)T_{q\ell}(z)\Delta H_{pq}(z)}{1 - T_{pq}(z)\Delta H_{pq}(z)}. \tag{3.36}$$

This is the expression that is desired. The transfer functions $T_{kp}(z)$, $T_{q\ell}(z)$, and $T_{qp}(z)$ in (3.36) are evaluated for the branch transmittance $H_{pq}(z)$ at its nominal value $H'_{pq}(z)$. The relation is valid for both incremental changes and large changes of $\Delta H_{pq}(z)$ and, therefore, is referred to as a large-change sensitivity relation. This relation is not new although the derivation is new and is useful for illustrating the connection between the incremental sensitivities and the large-change sensitivity. Various forms of this large-change sensitivity expression have been used extensively in feedback control theory for continuous time systems [25], [26]. It is sometimes referred to as the bilinear theorem.

The condition for which the infinite sum in (3.35) converges can be obtained by recognizing that it is a geometric series. Consequently, from the conditions for absolute convergence of a geometric series, we must require that

$$|T_{qp}(z)\ \Delta H_{pq}(z)| < 1. \tag{3.37}$$

If this condition is not satisfied, then the closed form

solution in (3.36) is not assured by the above proof. The proof can be extended, however, and the above condition (3.37) can be relaxed by applying the concept of analytic continuation. If $T_{qp}(z)$ and $\Delta H_{pq}(z)$ are analytic on and outside of the unit circle and the convergence condition (3.37) is satisfied on some circular contour, $|z| = R \geq 1$, in the z plane, then by the maximum modulus theorem (3.37) is satisfied for all z such that $|z| \geq R$. Furthermore, if we assume that the original network is stable, then $T_{kp}(z)$, $T_{q\ell}(z)$, and $T_{k\ell}(z)$ for $H_{pq}(z) = H'_{pq}(z)$ are analytic for $|z| \geq R \geq 1$. Consequently, the closed form expression (3.36) must be an analytic expression for $|z| \geq R$ because the numerator terms are analytic and the infinite sum is absolutely convergent over this region. By analytic continuation we can now extend relation (3.36) by allowing the radius R of the boundary of the region of analytic convergence to be reduced until the first pole is encountered. This pole can either be attributed to a pole of $T_{k\ell}(z)$, $T_{kp}(z)$, $T_{q\ell}(z)$, $\Delta H_{pq}(z)$ or a zero of the denominator expression

$$1 - T_{qp}(z)\Delta H_{pq}(z). \tag{3.38}$$

This last expression is often referred to in feedback control theory as the return-difference [25], [26]. If $\Delta H_{pq}(z)$ is allowed to vary starting from zero, then instability of the network will occur when a zero of expression (3.38) moves

across the unit circle. A useful and well-known test for this stability condition is the Nyquist stability test [26].

From the condition for absolute convergence (3.37), we can also establish a weaker condition for stability. If convergence of the sum in (3.35) exists on the contour $|z| = |e^{j\omega}| = 1$ and if the original network is stable, then we are assured that the new network with $H_{pq}(z) = H'_{pq}(z) + \Delta H_{pq}(z)$ is stable. That is, stability is guaranteed if

$$|T_{qp}(e^{j\omega})\Delta H_{pq}(e^{j\omega})| < 1 \tag{3.39}$$

or

$$|\Delta H_{pq}(e^{j\omega})| < \frac{1}{|T_{qp}(e^{j\omega})|} . \tag{3.40}$$

For an elementary network in which we are allowing only one coefficient to vary, $\Delta H_{pq}(z)$ will be of the form

$$H_{pq}(z) = \Delta c \tag{3.41}$$

or

$$H_{pq}(z) = \Delta d z^{-1} \tag{3.42}$$

for coefficient or coefficient and delay branches, respectively. The stability condition (3.40) then becomes

$$|\Delta c| < \frac{1}{|T_{qp}(e^{j\omega})|}$$

or

$$|\Delta d| < \frac{1}{|T_{qp}(e^{j\omega})|} ,$$

respectively. To guarantee stability we must require that the above conditions for absolute convergence are satisfied at all frequencies ω. Consequently, they become

$$|\Delta c| < \frac{1}{\max_{\omega} |T_{qp}(e^{j\omega})|} \tag{3.43}$$

and

$$|\Delta d| < \frac{1}{\max_{\omega} |T_{qp}(e^{j\omega})|} . \tag{3.44}$$

If $|\Delta c|$ or $|\Delta d|$ satisfy the above bounds, then we are guaranteed that the new network is stable. The converse is not true. That is, if $|\Delta c|$ or $|\Delta d|$ exceed the bounds (3.43) or (3.44), respectively, we cannot conclude that the network will necessarily be unstable. For this case, we must resort to the stronger Nyquist test.

3.8 A NETWORK SENSITIVITY PROPERTY FOR NONRECURSIVE COMPUTABLE NETWORKS

For the special case of nonrecursive computable network structures it has been shown in Theorem 2.3 that for elementary network, the matrix representation can be written in a form such that the matrices $\underline{H}_c^t$ and $\underline{H}_d^t$ are both zero on and

above their main diagonals simultaneously. It was further concluded that for the matrix in this form, the transpose of the transfer function matrix, $\underline{T}^t(z)$, is lower triangular. From these observations, it is then apparent that if a branch coefficient h_{pqc} or h_{pqd} is nonzero, then $T_{qp}(z)$ must necessarily be zero. Therefore, for nonrecursive computable network structures the large-change sensitivity relation (3.36) can be written in the form

$$T_{k\ell}(z)\Big|_{H_{pq}(z)=H'_{pq}(z)+\Delta H_{pq}(z)} = T_{k\ell}(z)\Big|_{H_{pq}(z)=H'_{pq}(z)} + (T_{kp}(z)\ T_{q\ell}(z))\Big|_{H_{pq}(z)=H'_{pq}(z)} \Delta H_{pq}(z). \qquad (3.45)$$

Furthermore, because $T_{qp}(z)$ is zero, the condition for absolute convergence (3.37) is guaranteed.

From relation (3.45) it can be observed that, for nonrecursive computable networks, the change in the system function with respect to the change in a branch coefficient is a linear function whose slope is given by the first-order derivative. Also, from (3.28) it is obvious that for this class of networks all higher-order derivatives for $n > 1$ must be zero because $T_{qp}(z)$ is zero.

3.9 HOMOGENEITY, SENSITIVITY, AND GROUP DELAY IN DIGITAL FILTERS

A number of interesting properties concerning first-order sensitivities of digital filters can be determined with the aid of a sensitivity relation associated with homogeneous functions [27]. A function of n parameters, $F(c_1, c_2, \ldots c_n)$, is defined to be homogeneous with respect to these parameters $c_i (i = 1, 2, \ldots n)$ if we can say that

$$F(\lambda c_1, \lambda c_2, \ldots \lambda c_n) = \lambda^M F(c_1, c_2, \ldots c_n) \tag{3.46}$$

where λ is an arbitrary number and M is defined as the order of homogeneity. The derivative of this function with respect to λ can be determined by means of the chain rule

$$\frac{\partial F(\lambda c_1, \lambda c_2, \ldots \lambda c_n)}{\partial \lambda} = \sum_{i=1}^{n} \frac{\partial F(\lambda c_1, \lambda c_2, \ldots \lambda c_n)}{\partial \lambda c_i} \frac{\partial \lambda c_i}{\partial \lambda}$$

$$= M\lambda^{M-1} F(c_1, c_2, \ldots c_n). \tag{3.47}$$

By recognizing that the derivative with respect to λ of λc_i is c_i and letting $\lambda = 1$ we get Euler's relation for homogeneous functions,

$$\sum_{i=1}^{n} c_i \frac{\partial F}{\partial c_i} = MF \tag{3.48}$$

where

$$F = F(c_1, c_2, \ldots c_n).$$

By dividing through by F it is apparent that this relation states that the sum of the relative sensitivities of a homogeneous function must equal the constant M, the order of homogeneity. That is,

$$\sum_{i=1}^{n} S_i^r[F] = \sum_{i=1}^{n} \frac{c_i}{F} \frac{\partial F}{\partial c_i} = M \tag{3.49}$$

where the relative sensitivity $S_i^r[F]$ of the function F with respect to a coefficient c_i is defined as

$$S_i^r[F] \triangleq \frac{\partial \ln F}{\partial \ln c_i} = \frac{c_i}{F} \frac{\partial F}{\partial c_i} . \tag{3.50}$$

For analog circuits it can be shown that the impedances and transfer functions are homogeneous functions with respect to their element values [28], [29], [30]. This property has led to a number of interesting sensitivity relations for analog networks [30]. In the case of digital networks, the transfer functions are, in general, not homogeneous with respect to their coefficients. However, some special classes of digital structures do exhibit such homogeneity. In particular, two such classes of structures are the direct form and the direct-canonic forms. For these two classes of structures the transfer function can be expressed as

$$T = T(z) = \frac{\sum_{i=0}^{n} a_i z^{-i}}{\sum_{i=0}^{m} b_i z^{-i}} . \qquad (3.51)$$

From (3.51) three forms of homogeneity can be observed. T is homogeneous of degree 1 with respect to the coefficients a_i. That is,

$$T(\lambda a_0, \lambda a_1, \ldots \lambda a_n) = \lambda^1 T(a_0, a_1, \ldots a_n). \qquad (3.52)$$

T is homogeneous of degree -1 with respect to the coefficients b_i (the unity scale factor b_0 must be included as a coefficient)

$$T(\lambda b_0, \lambda b_1, \ldots \lambda b_m) = \lambda^{-1} T(b_0, b_1, \ldots b_m). \qquad (3.53)$$

T is homogeneous of degree zero with respect to all of the coefficients a_i and b_i

$$T(\lambda a_0, \ldots \lambda a_n, \lambda b_0, \ldots \lambda b_m) = \lambda^0 T(a_0, \ldots a_n, b_0, \ldots b_m). \qquad (3.54)$$

From these homogeneous properties we can state with the aid of (3.49) that, for direct form and direct-canonic form structures, the sum of the relative sensitivities of the coefficients a_i is equal to 1, the sum of the relative

sensitivities of the coefficients b_i is equal to -1 and the sum of the relative sensitivities with respect to all of the coefficients (a_0 and b_0 must be included) is equal to zero.

Another class of structures which can exhibit homogeneity with respect to the coefficients are the structures which are derived by continued fraction expansions [31].

A more general application of homogeneity for digital structures can be observed when the complex frequency z is included as a parameter. In this case, it can be seen from the matrix representation (see Chapter 2),

$$\underline{Y}(z) = \underline{H}_c^t \underline{Y}(z) + \underline{H}_d^t \underline{Y}(z) z^{-1} + \underline{U}(z), \tag{3.55}$$

and the matrix of system functions,

$$\underline{T}(z) = (\underline{I} - \underline{H}_c - \underline{H}_d z^{-1}), \tag{3.56}$$

that the transfer functions of a digital network are homogeneous of degree zero with respect to the coefficients of the delay branches $\underline{H}_d$ and the complex frequency z. That is, for every element T in $\underline{T}$ we can write

$$T(\lambda d_1, \lambda d_2, \ldots, \lambda d_m, \lambda z) = \lambda^0 T(d_1, d_2, \ldots, d_m, z) \tag{3.57}$$

where d_i (i = 1, 2, ..., m) corresponds to all of the nonzero coefficients in $\underline{H}_d$. Because of this homogeneity property, we

can apply relation (3.49) to obtain

$$\frac{z}{T}\frac{\partial T}{\partial z} + \sum_{i=1}^{m} \frac{d_i}{T}\frac{\partial T}{\partial d_i} = M = 0. \tag{3.58}$$

The first term in (3.58) corresponds to the relative sensitivity of T with respect to frequency and the m terms of the summation correspond to the relative sensitivities of T with respect to the coefficients in $\underline{H}_d$. Consequently, (3.58) relates the frequency sensitivity of T to specific coefficient sensitivities of T.

For the case where $z = e^{j\omega}$, we can obtain from (3.58) expressions relating the frequency sensitivity of the natural logarithm of the magnitude of the system function and the group delay to the coefficient sensitivities. To show this, it is first necessary to define some additional terms. For $z = e^{j\omega}$, we can write the system function T in polar form as

$$T = |T|e^{j\theta} \tag{3.59}$$

where $|T|$ is the magnitude of the system function and θ is the phase. Both are functions of the frequency ω. The group delay of the system can now be defined as

$$\tau = \text{group delay of } T \triangleq -\frac{\partial \theta}{\partial \omega}. \tag{3.60}$$

As the frequency parameter of interest is ω rather than z, we can express the sensitivity of T with respect to ω, in

terms of the sensitivity with respect to z, by applying the chain rule. This gives

$$\frac{\partial T}{\partial \omega} = \frac{\partial T}{\partial e^{j\omega}} \cdot \frac{\partial e^{j\omega}}{\partial \omega} = j e^{j\omega} \frac{\partial T}{\partial e^{j\omega}} = jz \left.\frac{\partial T}{\partial z}\right|_{z = e^{j\omega}} . \tag{3.61}$$

Dividing by T and j gives

$$\frac{z}{T} \left.\frac{\partial T}{\partial z}\right|_{z=e^{j\omega}} = \frac{-j}{T} \frac{\partial T}{\partial \omega} . \tag{3.62}$$

This expression exactly corresponds to the first term in (3.58) for $z = e^{j\omega}$. By substituting this expression into (3.58) and applying the relative sensitivity definition (3.50) to the m terms in the summation, we get

$$\frac{-j}{T} \frac{\partial T}{\partial \omega} + \sum_{i=1}^{m} S_i^r[T] = 0$$

or

$$\frac{1}{T} \frac{\partial T}{\partial \omega} = -j \sum_{i=1}^{m} S_i^r[T]. \tag{3.63}$$

We can further recognize that the right term of (3.63) can be written in the form

$$\frac{1}{T} \frac{\partial T}{\partial \omega} = \frac{\partial \ln T}{\partial \omega} = \frac{\partial \ln |T|}{\partial \omega} + j \frac{\partial \theta}{\partial \omega} . \tag{3.64}$$

Substituting this into (3.63) gives

$$\frac{\partial \ln |T|}{\partial \omega} + j \frac{\partial \theta}{\partial \omega} = - j \sum_{i=1}^{m} S_i^r[T]. \tag{3.65}$$

As $|T|$, θ, and ω are real variables, the terms $(\partial \ln |T|)/\partial\omega$ and $\partial\theta/\partial\omega$ are real. Thus, the two terms on the left of (3.65) correspond to the real and imaginary parts, respectively. By separating the real and imaginary parts of (3.65), we can get the desired relations. The real part gives the sensitivity of the natural logarithm of the magnitude of the system function, in terms of the imaginary parts of the relative coefficient sensitivities, as

$$\frac{\partial \ln |T|}{\partial \omega} = \text{Im} \sum_{i=1}^{m} S_i^r[T] = \sum_{i=1}^{m} \text{Im}\, S_i^r[T]. \tag{3.66}$$

The imaginary part of (3.65), together with the aid of (3.60) relates the group delay, τ, to the sum of the real parts of the relative coefficient sensitivies as

$$\tau = - \frac{\partial \theta}{\partial \omega} = \text{Re} \sum_{i=1}^{m} S_i^r[T] = \sum_{i=1}^{m} \text{Re}\, S_i^r[T]. \tag{3.67}$$

The coefficient sensitivities in (3.65) and (3.66) correspond to all of the nonzero coefficients in $\underline{H}_d$.

In this chapter we have presented a number of basic theorems and properties relating to sensitivity. In Chapter 5

we will show how some of these theorems and properties can be applied to develop efficient computer-aided network analysis techniques for the analysis of arbitrary digital networks.

Chapter IV

PRECEDENCE RELATIONS AND PARALLELISM IN THE IMPLEMENTATION OF DIGITAL FILTERS

4.1 INTRODUCTION

In the implementation of a digital filter we are faced with many design tradeoffs, especially if we intend to implement the filter as a special-purpose piece of hardware. Of particular importance is the choice of an appropriate filter structure. This choice involves many considerations. If we are interested in speed as one criterion, then we may desire a structure that can be computed with as much parallelism in the hardware as possible. The topology of a structure will basically determine how much computation can be done in parallel and how much must be done serially. It also determines the tradeoffs available in situations where we have a restricted amount of hardware and want to do part of the filter computations in parallel and part of the computations serially. In this chapter we address ourselves to these problems.

We will assume in this chapter, without any loss in generality, that all coefficient and delay branches in a structure have unit coefficients. We can always do this by representing such branches as a cascade of a coefficient branch and a (unity-gain) delay branch.

4.2 PRECEDENCE RELATIONS IN FILTER STRUCTURES

In Chapter 2, Section 2.5, we have established some fundamental topological constraints which must exist in a filter structure in order for it to be computable. In particular, we have established that, for a computable network, the matrix representation can be written in a form such that $\underline{H}_c^t$ is zero on and above the main diagonal. Furthermore, we have shown in the proof of Theorem 2.1 that for a computable network we can draw the coefficient branch structure of the network in the form depicted by Fig. 4.1.

An algorithm for determining the coefficient branch structure of a computable network in the form illustrated by Fig. 4.1 is given in the proof of Theorem 2.1. It can be restated as follows: Starting with the initial network, we extract all nodes in the network which do not have coefficient branches entering them. We assign these nodes to set $\{n_1\}$ and erase all of the nodes $\{n_1\}$ and the branches connecting these nodes from the structure. We then repeat the above procedure on the remaining structure successively generating node sets $\{n_2\}$, $\{n_3\}$, ..., etc., until all of the nodes in the network have been removed. Knowing the node sets $\{n_1\}$, $\{n_2\}$, ..., $\{n_f\}$, we can then go back to the initial network and construct the coefficient branch topology as it is illustrated in Fig. 4.1. If, at any stage in the process, we cannot find at least one node in the remaining network which does not have a coefficient branch entering it, then the

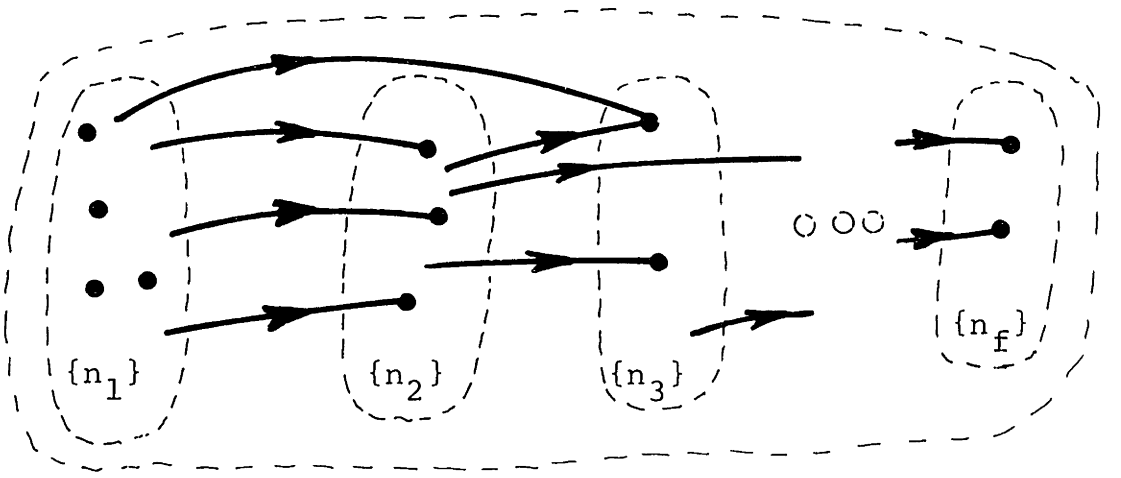


Fig. 4.1 Coefficient branch structure of a computable network.

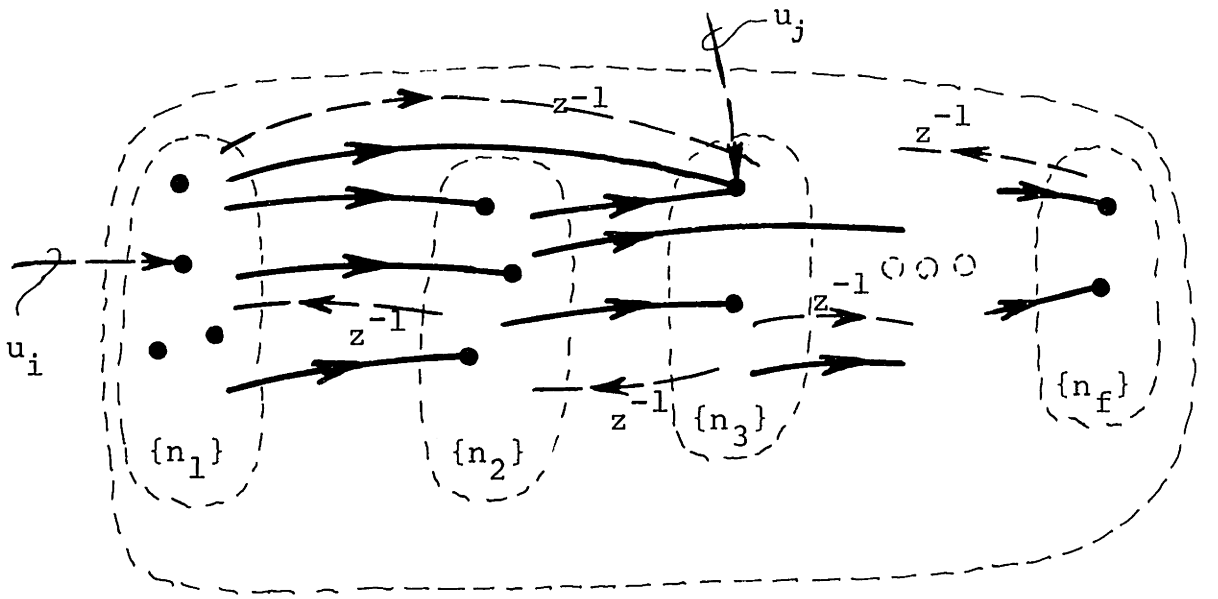


Fig. 4.2 Illustration of the topology of a computable network in precedence form.

remaining nodes must be associated with loops and the network is noncomputable.

Once we have the coefficient branch structure of a network in the above form, we can then go back and insert the source branches and (unity-gain) delay branches as illustrated in Fig. 4.2. This form of the network structure will be referred to as a precedence form of the network topology as it is useful for illustrating the basic precedence of operations that must be observed in the computation of the filter.

If we think of the computations in a digital structure in terms of the evaluations of node variables, then we can observe, from the precedence form of the structure, the maximum amount of parallelism that can be achieved in the structure. This can be accomplished by computing, simultaneously, all N_1 node values in node set $\{n_1\}$, then all N_2 node values in node set $\{n_2\}$, etc., until we have consecutively computed all f node sets. To avoid conflict, the node sets must be computed serially. Thus, if we consider the evaluation of node values as our basic filter operations, the precedence graph not only tells us the maximum amount of parallelism that can be achieved in the structure, but also the minimum amount of serialism that is inherent in the structure and cannot be avoided. Unfortunately, this is a hypothetical oversimplified situation because in practice, we want to consider multiplies and adds as the fundamental operations rather than node evaluations. This introduces several

complications to the above concepts of parallelism and serialism. When we consider multiplies, we do not want to include those coefficients that correspond to unity gains and we may not want to consider coefficients such as 1/2 or 2 as they can be more easily implemented as shift operations. If a node does not have more than one branch entering it, we do not have to perform an addition to obtain the node value. Alternatively, if a node has more than two branches entering it, we are faced with the problem of doing more than one addition to obtain the node value (assuming that we can add only two numbers at a time) and we must decide which adds are to be performed first. While these issues alter the amount of parallelism or serialism that can be achieved in a hardware implementation, we must still obey the precedence relations dictated by the precedence form of the network.

4.3 PARALLELISM AND SERIALISM TRADE-OFFS WITH RESPECT TO MULTIPLIERS

In a hardware implementation of a digital filter we often perform the multiplies as a series of additions. Consequently, the time needed to perform a multiply is often much greater than the time needed to perform an add or other operations in the filter. In this case, we can approximate the time required to compute one filter cycle in terms of multiplier cycle times. For this situation, we can assume that the fundamental tradeoffs and limitations between parallelism and

serialism (e.g. speeds vs. cost) in hardware are determined by the multiplier precedence relations.

To obtain the multiplier precedence relations of a structure, we must first determine the coefficient precedence relations. As we have assumed that the coefficients are only associated with coefficient branches, we can obtain the coefficient precedence relations from the coefficient branch topology in Fig. 4.1. To do this, we recognize that all of the multiplies associated with coefficient branches leaving nodes in set $\{n_1\}$ can be performed simultaneously (see Fig. 4.3). We assign these coefficients to set $\{c_1\}$. Once the multiplies in this set have been performed, we can compute all of the multiplies in $\{c_2\}$, etc., until we have performed all f - 1 sets of multiplies. If any of the coefficients correspond to factors of 1, 1/2, 2, etc., we may not want to implement them as hardware multiplies (1/2 and 2 can be performed by 1 bit right or left shifts). Therefore, if a coefficient in $\{c_2\}$, for example, is preceded only by these types of branches in $\{c_1\}$, then presumably it can be implemented as a hardware multiply in approximately the same time slot as the multiplies in $\{c_1\}$. Consequently, we can alter the precedence relations in Fig. 4.3 to account for these situations.

From the coefficient precedence relations in Fig. 4.3, we can draw a multiplier precedence graph for which the multiplies correspond only to the coefficients in the network which must be realized as hardware multiplies. This

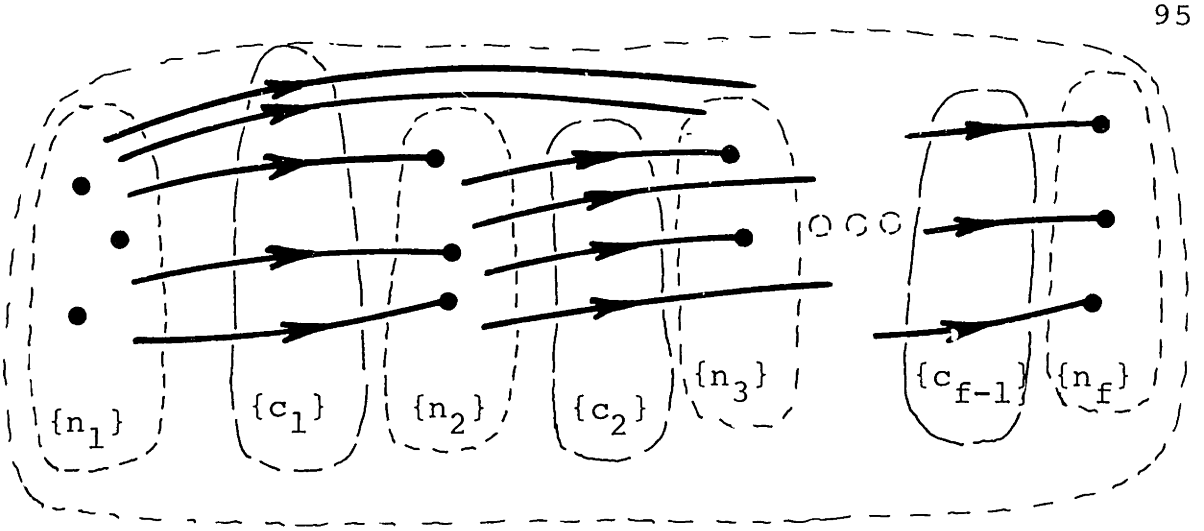


Fig. 4.3 Illustration of the precedence of coefficients.

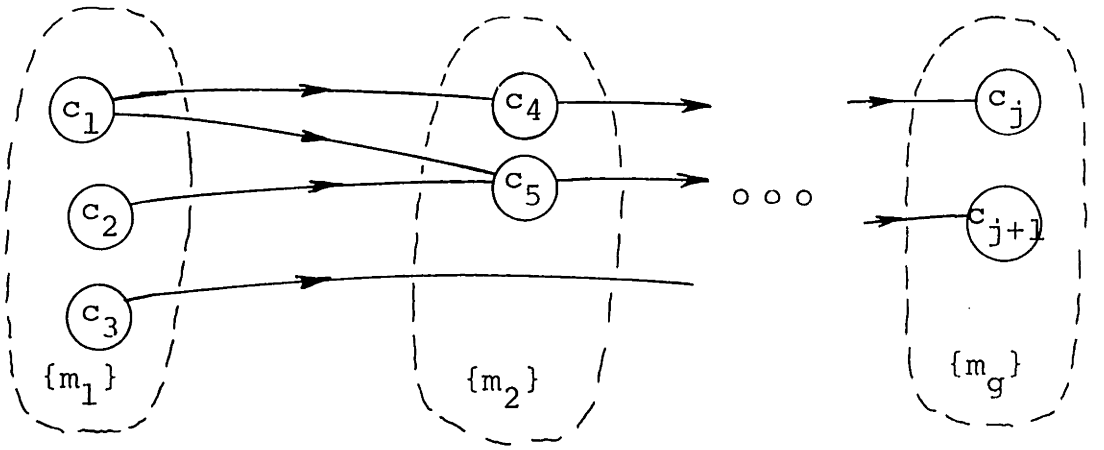
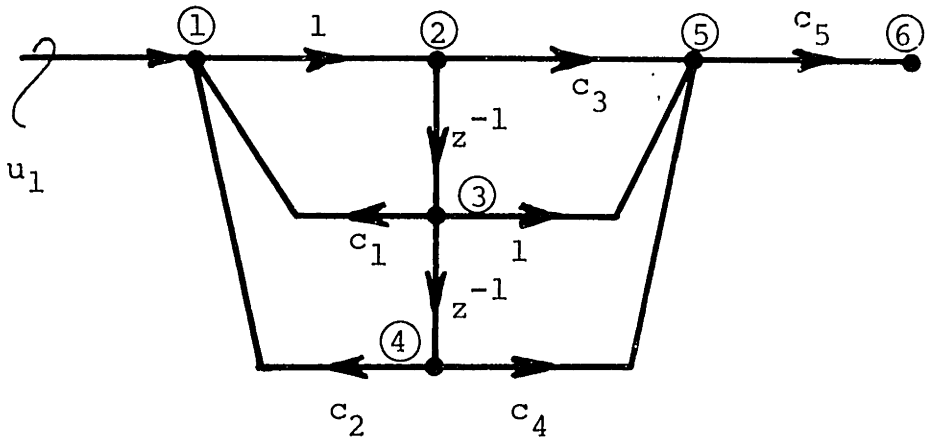


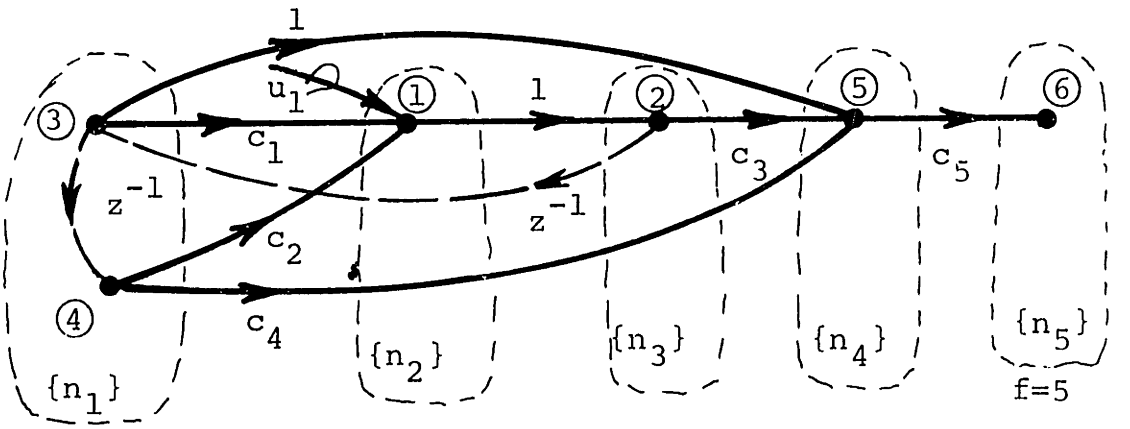
Fig. 4.4 An example of a multiplier precedence graph.

precedence graph is illustrated in Fig. 4.4. In this graph, circles are used to indicate the multiply operations and the arrows merely indicate the precedence relations between these multiplies. For example, in the graph of Fig. 4.4 the multiply associated with coefficient c_5 cannot be performed until the multiplies associated with coefficients c_1 and c_2 have been performed. Only the required multiply operations in the filter and their precedence relations are depicted in this graph. The multiplies associated with set $\{m_1\}$ can all be performed simultaneously without conflict in approximately the same time slot. Once these multiplies have been performed, all of the multiplies in set $\{m_2\}$ can be performed simultaneously in the next time slot, etc.. Coefficients in set $\{m_1\}$ correspond to all of the coefficients in $\{c_1\}$ in Fig. 4.3 which must be realized as hardware multiplies; plus all of the coefficients in sets $\{c_2\}$, $\{c_3\}$, ..., $\{c_{f-1}\}$, which are only preceeded by coefficients such as 1, 1/2, and 2, and can, therefore, be implemented in the first multiplier time slot. Coefficients in set $\{m_2\}$ correspond to those remaining coefficients in set $\{c_2\}$ which cannot be moved to $\{m_1\}$ plus any coefficients in sets $\{c_3\}$, $\{c_4\}$, ..., $\{c_{f-1}\}$ which can be moved up without conflict in the multiplier precedence relations. By continuing this procedure we can generate the multiplier precedence graph illustrated in Fig. 4.4 where, obviously, $g \leq f - 1$.

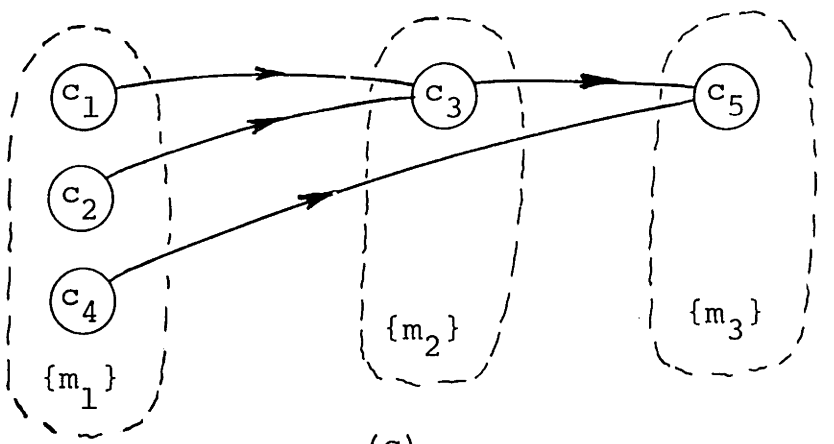
As an example, consider the network in Fig. 4.5(a).



(a)



(b)



(c)

Fig. 4.5 (a) An example of a digital network,

(b) The precedence form of the network,

(c) The multiplier precedence graph of the network.

The precedence form of this network is illustrated in Fig. 4.5(b) and the multiplier precedence graph is given in Fig. 4.5(c). From the precedence form of the network (Fig. 4.5 (b)), we can see that there are five inherent levels of serialism with respect to the nodes in the structure. From the multiplier precedence graph (Fig. 4.5(c)), we can see that there are three inherent levels of serialism in the computations of the multiplies. Consequently, if we implement this network with the maximum amount of parallelism, the minimum computation time which can be achieved for one filter cycle computation is approximately three multiplier cycle times.

The multiplier precedence graph is a useful device for analyzing the tradeoffs between the number of multipliers M, that we use in a particular hardware realization of a structure and the minimum computation time in multiplier cycles, t_m, that it takes to compute one filter cycle. One way to illustrate this tradeoff is to plot t_m against M. Consider the multiplier precedence graph in Fig. 4.5(c). If M = 1, then all of the multiplies must be computed serially. As there are five multiplies in the graph, $t_m = 5$. If M = 2, then we can perform the multiplies c_1 and c_2 in the first time slot, c_3 and c_4 in the second time slot, and c_5 in the third time slot. Consequently, $t_m = 3$. As this is the inherent level of serialism in this particular structure, we cannot do better than this by using more multipliers in the hardware

realization. The plot of t_m versus M for this structure is given in Fig. 4.6.

It is interesting to compare the t_m versus M plot for the above structure against other possible structures that can realize the samc overall system function. In particular, we can consider two extremes. A structure will be defined to be completely serial with respect to its multiplies if every level $\{m_i\}$ in its multiplier precedence graph contains only one possible multiply as illustrated in Fig. 4.7(a). Alternatively, a structure will be defined to be completely parallel with respect to its multiplies if its multiplier precedence graph contains only one level $\{m_1\}$ as indicated by Fig. 4.7(b). Plots of the t_m versus M for these two extremes and for the network of Fig. 4.5 are given in Fig. 4.8.

For a completely serial structure, we can observe that the t_m versus M plots correspond to the relation

$$t_m = C \tag{4.1}$$

where C is the total number of multiplies that must be performed in the structure. For a completely parallel structure, this relation is

$$t_m = \text{Int.} \left[\frac{C}{M}\right], \tag{4.2}$$

where Int. [•] corresponds to the operation of rounding up the number in brackets to the next largest integer. As can be

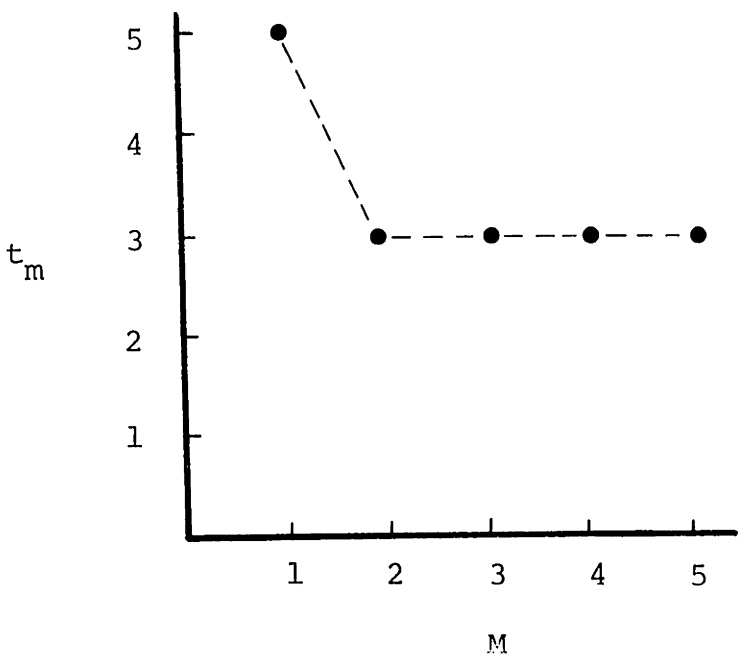


Fig. 4.6 Minimum multiplier cycle time, t_m, versus the number of multiplies, M, for the structure of Fig. 4.5

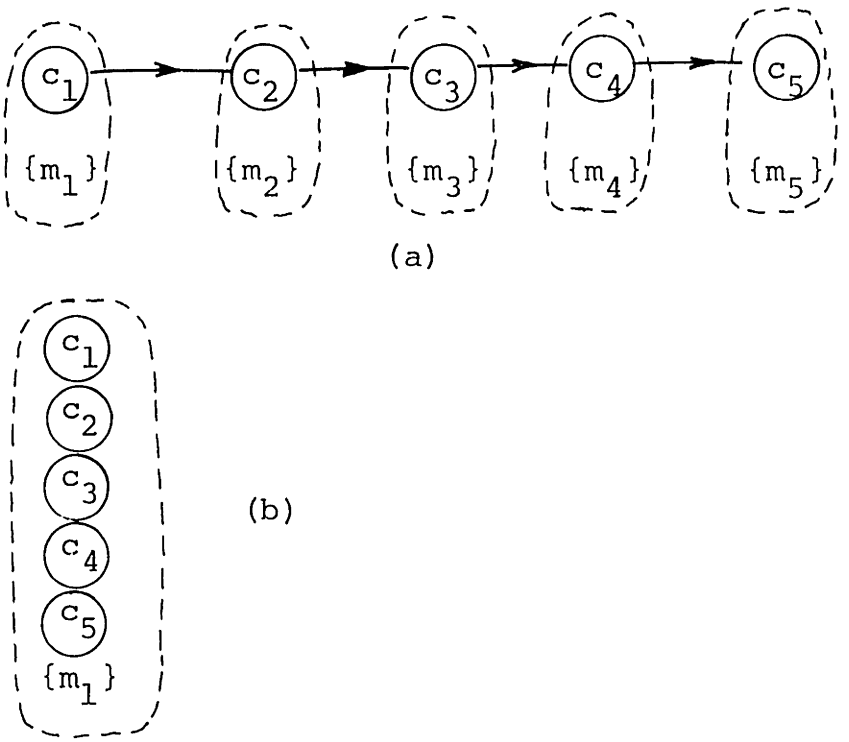


Fig. 4.7 Examples of multiplier precedence graphs that are (a) completely serial, and (b) completely parallel with respect to multiplies

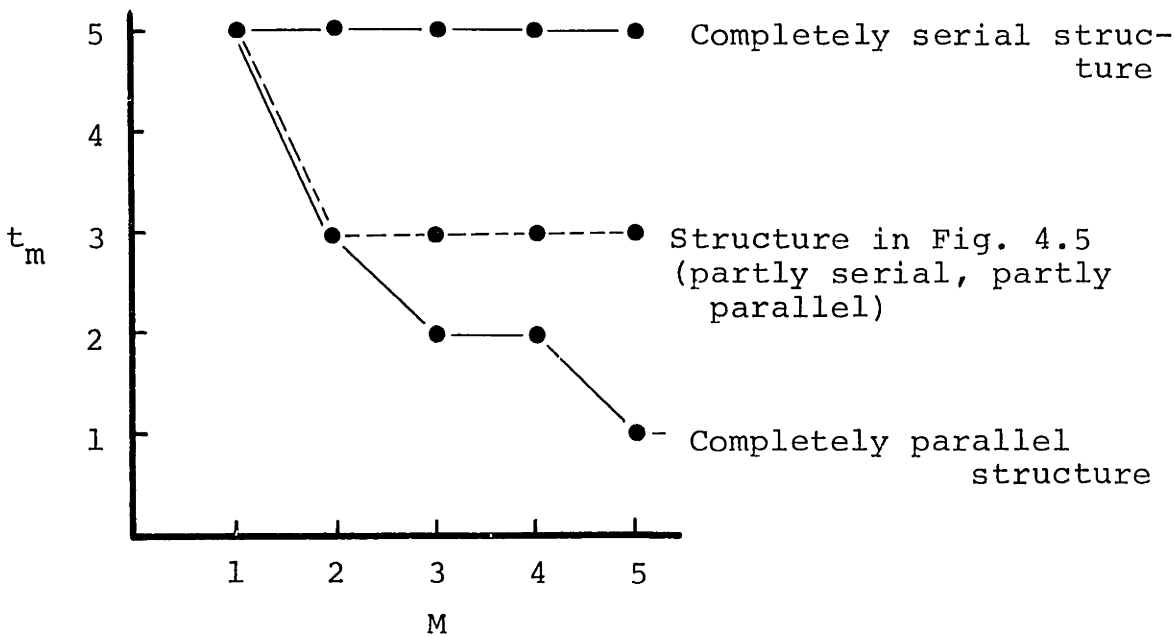


Fig. 4.8 Comparison between the t_m versus M graphs for completely serial and completely parallel structures.

seen in Fig. 4.8, many practical structures may have t_m versus M plots that lie somewhere between these two limits.

4.4 INCREASING PARALLELISM AND REDUCING SERIALISM IN STRUCTURES BY PIPELINING

In some cases the parallelism in a structure can be increased if extra delay in the system function can be tolerated. For example, the structure in Fig. 4.9 can be computed in approximately one multiplier cycle using two multipliers by performing the multiply c_1 for time index k at the same time that the multiply c_2 is being performed for time index k-1. Although there is a time delay of one sample through the network, the effective time to compute one filter cycle using two multipliers appears to be one multiplier cycle time instead of two as we would have expected using the concepts in the previous section. This technique of processing is often referred to as pipeline processing, and it appears, at first, that it contradicts our previous concepts of inherent serialism in structures. It does not violate these concepts, however, because the structure which is actually being computed by this technique is not the one in Fig. 4.9 but rather the structure in Fig. 4.10, and the coefficients c_1 and c_2 are actually being performed for the same time index. It is easy to see that the inherent serialism with respect to multipliers for this circuit is one. In effect, by using the pipeline scheme we have merely converted the structure of Fig.

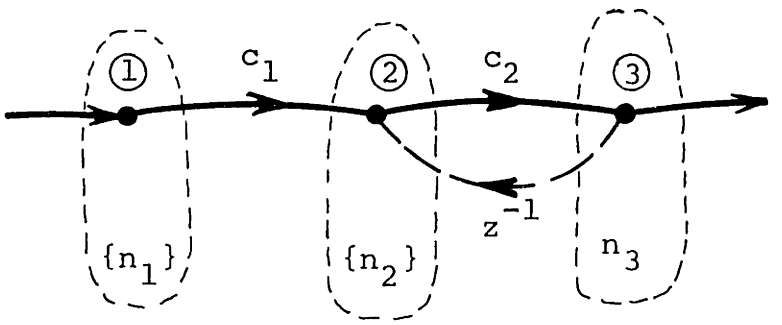
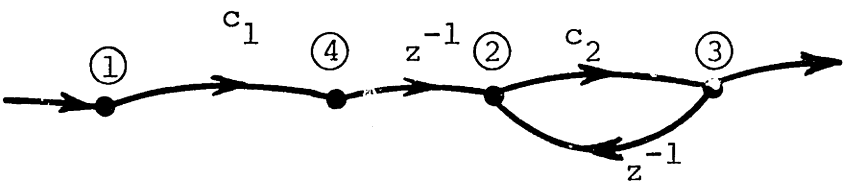
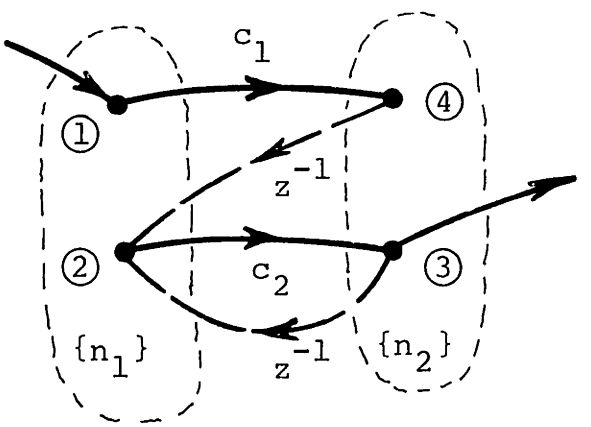


Fig. 4.9 A simple digital network with an inherent serialism of 2 with respect to multiplies.



(a)



(b)

Fig. 4.10 (a) modification of the circuit in Fig. 4.9 to increase parallelism and reduce serialism.

(b) The precedence form of the network of part (a).

4.9 to that of Fig. 4.10 and none of the concepts previously discussed have been violated.

We can think of pipelining as a process by which we add extra unit delays in appropriate parts of a structure in order to reduce its inherent serialism at the expense of introducing extra delay in certain transfer functions of the structure. These delays can only be inserted in those parts of a structure which are not associated with loops or feedback paths in the structure. Also, if a structure contains several feed-forward paths in parallel and a unit delay is added in one of these paths then a delay must also be added in each of the paths which are in parallel with it in order to preserve the feed-forward cancellations in the structure.

As an example, in the network of Fig. 4.5 we can reduce the amount of serialism by inserting a delay in series with the branch from node 5 to node 6. We can also reduce the serialism of this network further if we insert delays in series with the branches from nodes 2 to 5, 3 to 5, and 4 to 5. As these three branches are in parallel, delays must be inserted in all of them. By this technique we can reduce the amount of serialism with respect to multiplies for this circuit from 3 for the original structure to 1 for the pipelined structure (see Fig. 4.11).

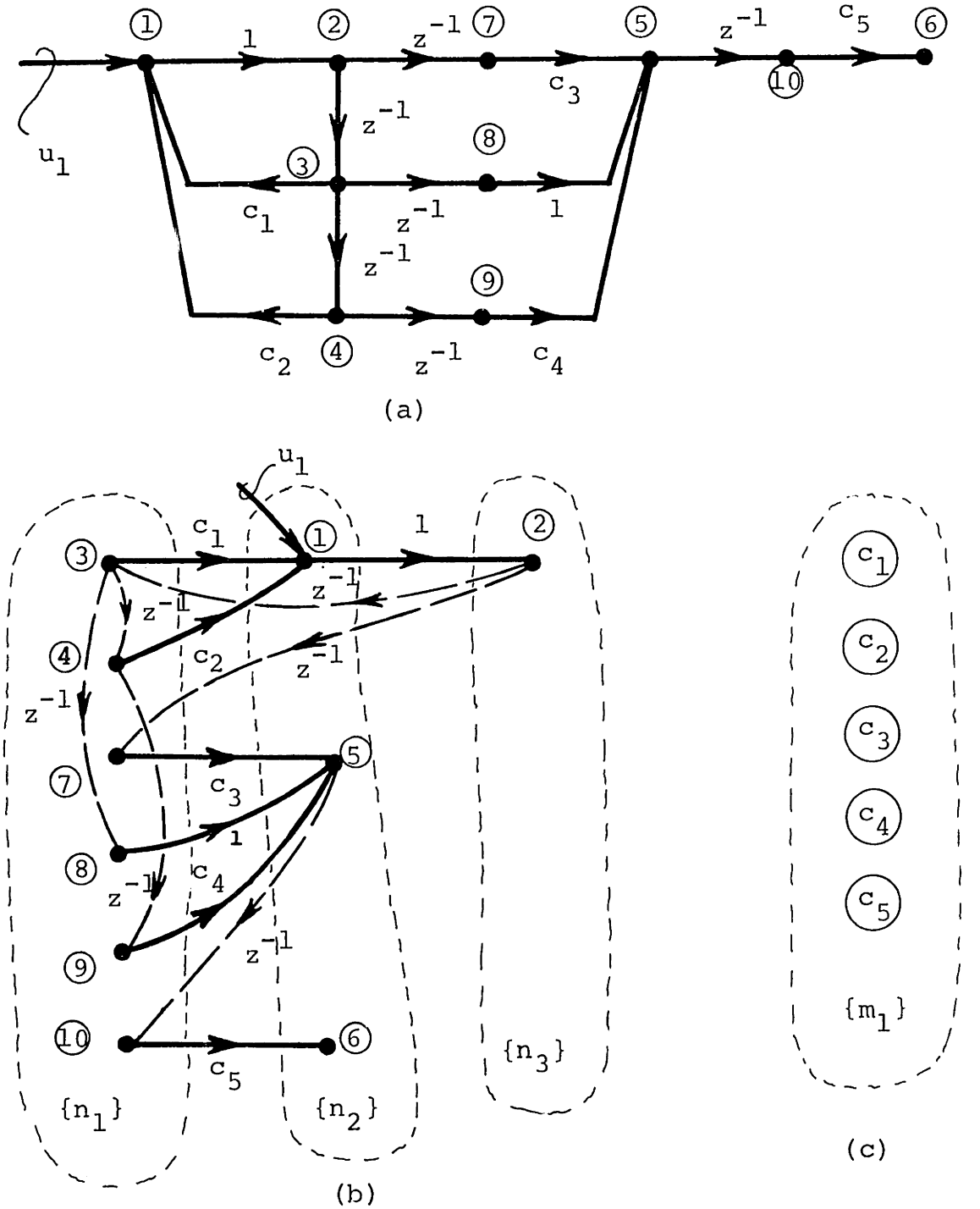


Fig. 4.11 (a) A pipelined form of the network in Fig. 4.5, (b) its precedence form, and (c) its multiplier precedence graph.

4.5 A COMPARISON OF RECURSIVE STRUCTURES IN TERMS OF PARALLELISM WITH RESPECT TO MULTIPLIERS

In this section we analyze some proposed filter structures on the basis of their inherent parallelism and serialism. From this comparison, we can draw conclusions as to what types of structures we may want to consider for applications where a high degree of parallel processing is required. The comparison is made for an arbitrary fourth-order system function but the results are extended to all orders of system functions. In Chapter 7 further comparisons are made for a bandpass system function in which the zeros are located on the unit circle and the number of coefficients in some of the structures can be reduced.

The details of synthesis of the various structures will not be given. All of the structures are canonic with respect to delays (4 delays) and multiplies (9 multiplies) with the exception of Example 4.

4.5.1 Common Digital Filter Structures

Three basic classes of structures have been initially considered for the realization of recursive digital filters [1], [2], [3], [4]. They are the direct form structures, parallel form structures, and the cascade structures with first- and/or second-order direct form sections.

Example 1 - Direct Form II (Canonic) Structures

The direct form II structure for an arbitrary fourth-order

system function is given in Fig. 4.12. The system function of this structure can be expressed as

$$T(z) = k \cdot \frac{1 + c_2 z^{-1} + c_4 z^{-2} + c_6 z^{-3} + c_8 z^{-4}}{1 - c_1 z^{-1} - c_3 z^{-2} - c_5 z^{-3} - c_7 z^{-4}},$$

where the coefficients in the numerator and denominator polynomials correspond to the coefficients in the structure, respectively. From the multiplier precedence graph of this structure (see Fig. 4.13), we can see that this class of structures is completely parallel with respect to the multiplies. This holds for all filter orders.

Example 2 - Parallel Form Structures

The class of parallel form structures is synthesized by expressing the system function in the form of a partial-fraction expansion. This leads to the structure in Fig. 4.14 and the system function can be expressed as

$$T(z) = k + c_1 \cdot \frac{1 + c_3 z^{-1}}{1 - c_2 z^{-1} - c_4 z^{-2}} + c_5 \cdot \frac{1 + c_7 z^{-1}}{1 - c_6 z^{-1} - c_8 z^{-2}}.$$

This class of structures is completely parallel with respect to its multiplies for all filter orders.

Example 3 - Cascade (Direct Form II) Structures

This class of structures is synthesized by factoring the system function into ratios of first- and/or second-order

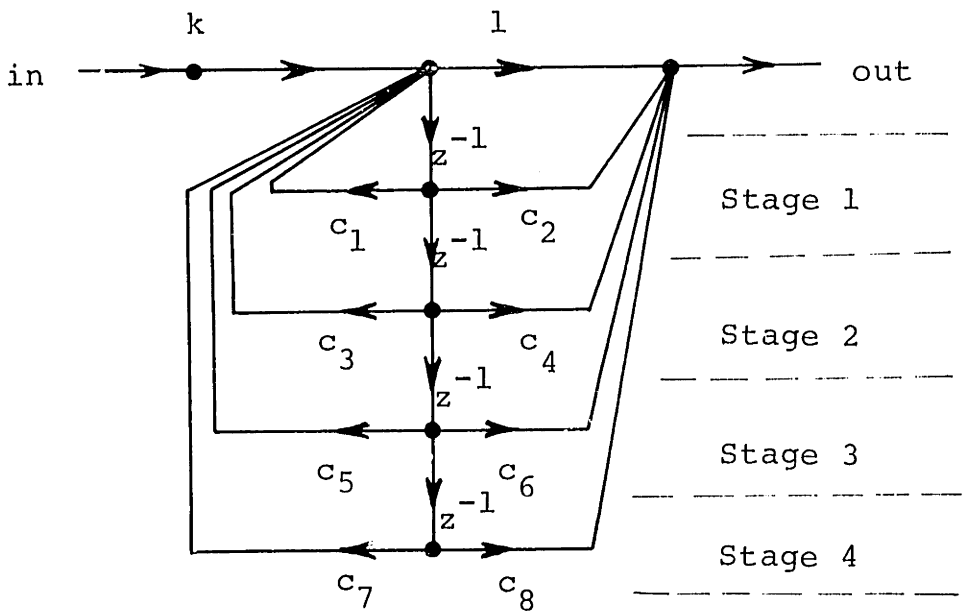


Fig. 4.12 Direct form II (canonic) structure for the realization of an arbitrary fourth-order system function, (example 1).

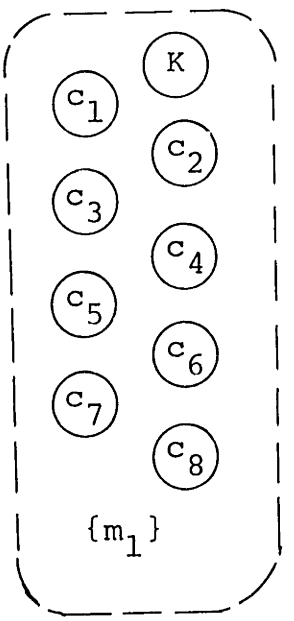


Fig. 4.13 Multiplier precedence graph for the structures in Examples 1, 2, and 3.

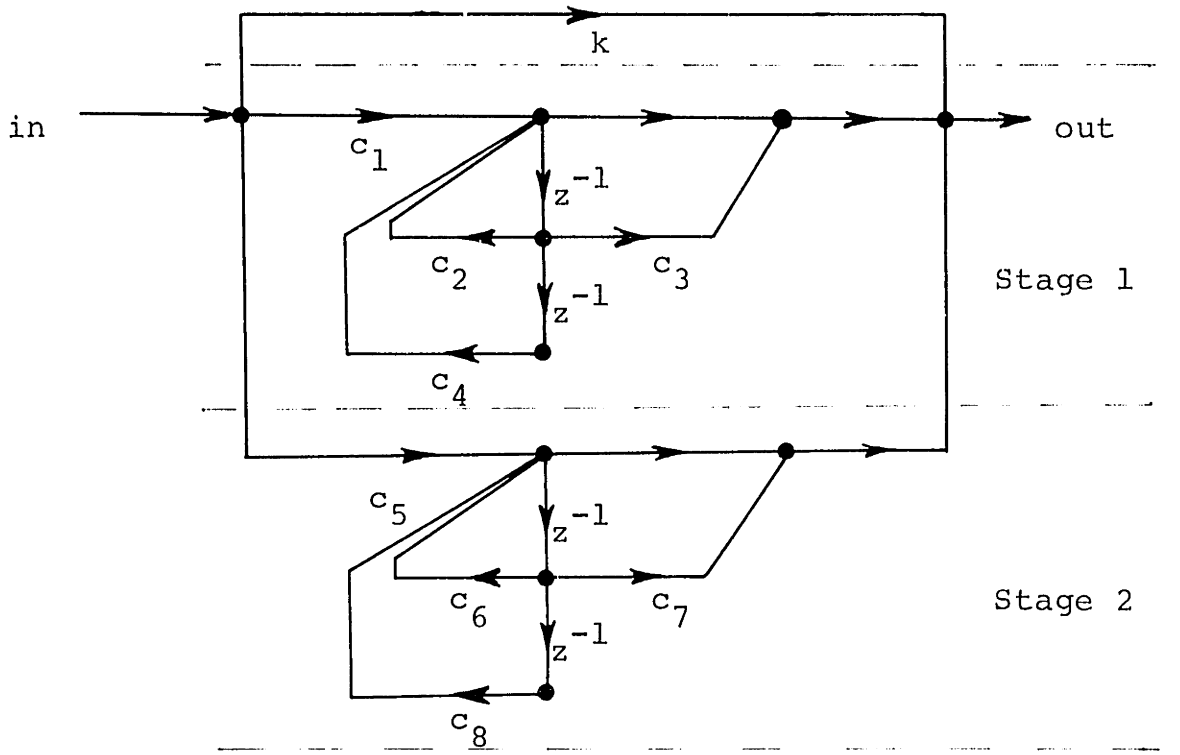


Fig. 4.14 Parallel form structure (Example 2).

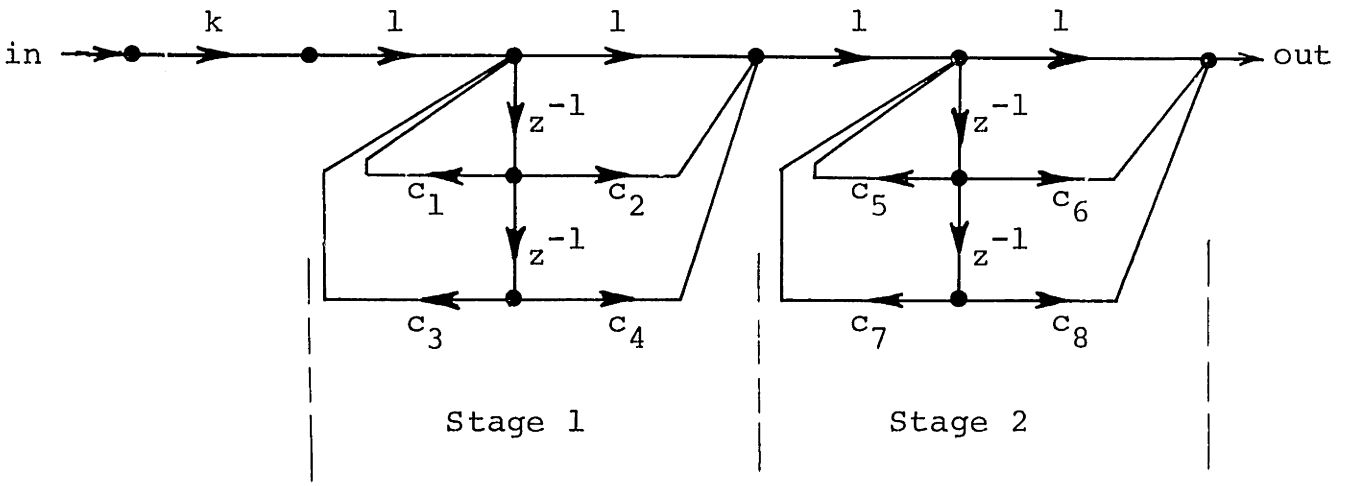


Fig. 4.15 Cascade (direct form II) structure (Example 3).

polynomials where first-order polynomials are chosen for real roots and second-order polynomials are chosen for complex conjugate root pairs. The structure is then realized as a cascade of first- and/or second-order direct form II sections (see Fig. 4.15). The system function for this structure is expressed as

$$T(z) = k \cdot \frac{1 + c_2 z^{-1} + c_4 z^{-2}}{1 - c_1 z^{-1} - c_3 z^{-2}} \cdot \frac{1 + c_6 z^{-1} + c_8 z^{-2}}{1 - c_5 z^{-1} - c_7 z^{-2}}$$

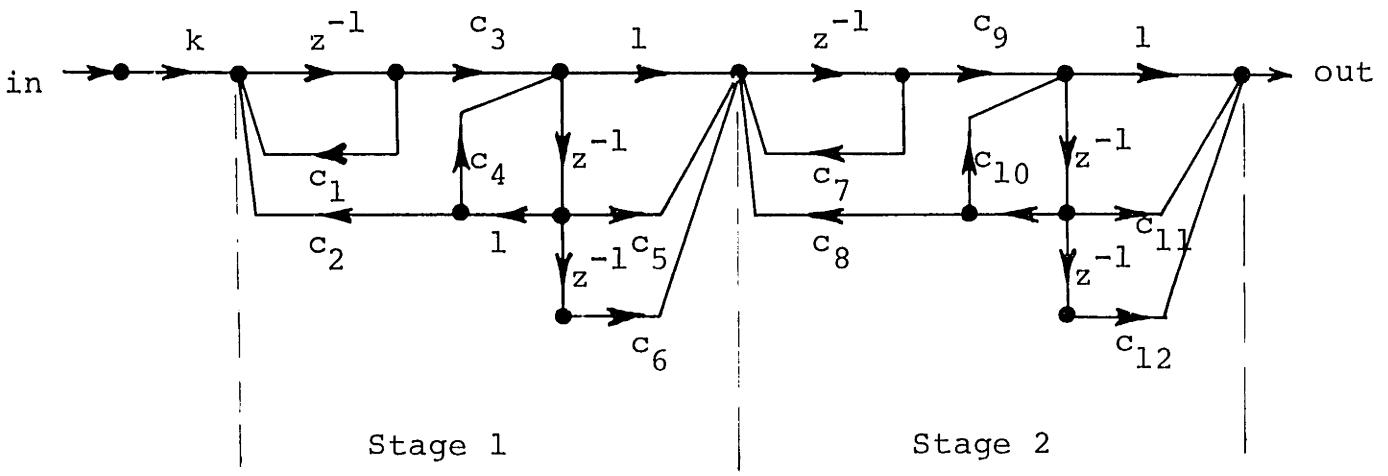
This class of structures is also completely parallel with respect to multiplies for all orders.

4.5.2 Cascade Structures

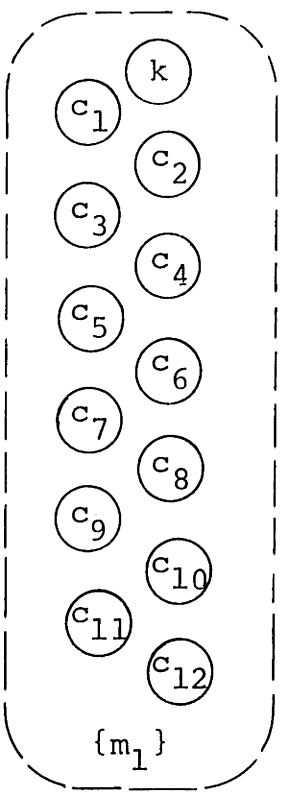
Instead of choosing direct form II sections for the second-order sections of a cascade structure, many other configurations for these second-order sections can be considered [2], [3], [32], [33], [34]. Some examples are analyzed here.

Example 4 - Cascade (Coupled Form) Structures

The cascade structure with coupled form second-order sections [2] is illustrated in Fig. 4.16(a). The system function can be expressed as



(a)



(b)

Fig. 4.16 (a) Cascade (coupled form) structure and (b) its multiplier precedence graph (example 4).

$$T(z) = k \cdot \frac{c_3 z^{-1}(1+c_5 z^{-1}+c_6 z^{-2})}{1 - (c_1+c_4) z^{-1} + (c_1 c_4 - c_2 c_3) z^{-2}} \times$$

$$\frac{c_9 z^{-1}(1+c_{11} z^{-1}+c_{12} z^{-2})}{1 - (c_7+c_{10}) z^{-1} + (c_7 c_{10} - c_8 c_9) z^{-2}} .$$

This class of structures is completely parallel with respect to its multiplies (see Fig. 4.16(b)) for all orders but it is not canonical in terms of these multiplies. Two extra multiply operations per stage must be performed in addition to that of the canonic structures. In addition, a one-sample delay is introduced at each stage thus producing an additional linear phase shift in the system response.

Example 5 - Cascade (Avenhaus B) Structures

Some interesting second-order canonic sections for cascade realizations have recently been proposed by Avenhaus [34]. Using circuit B (see [34]), we get the structure in Fig. 4.17(a). The system function for this class of structures can be expressed as

$$T(z) = k \cdot \frac{1 + c_1 z^{-1} + c_2 z^{-2}}{1 - (c_1 c_4 + c_2 c_3) z^{-1} - c_2 c_4 z^{-2}} \times$$

$$\frac{1 + c_5 z^{-1} + c_6 z^{-2}}{1 - (c_5 c_8 + c_6 c_7) z^{-1} - c_6 c_8 z^{-2}} .$$

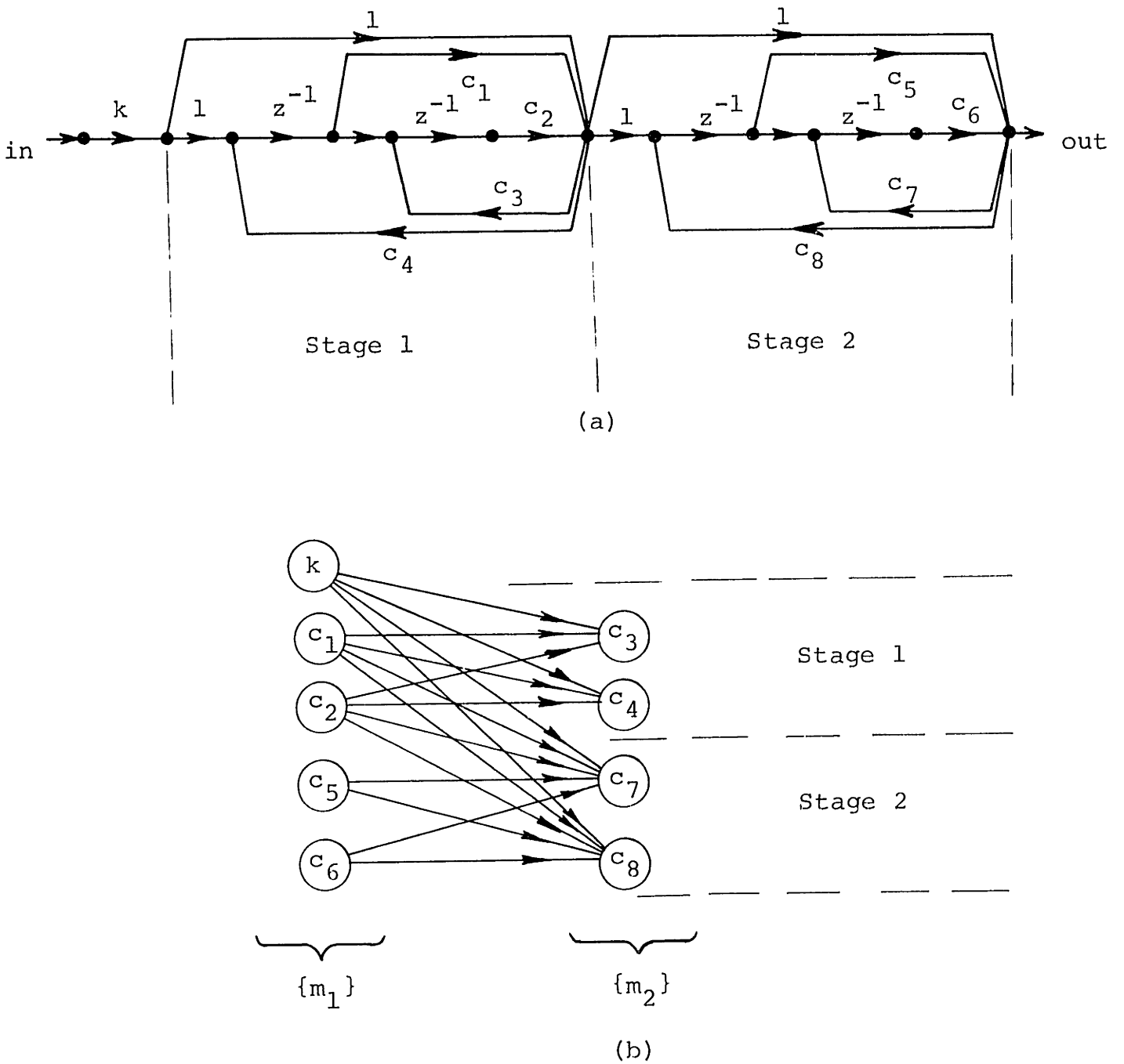


Fig. 4.17 (a) Cascade (Avenhaus B) structure and
(b) its multiplier precedence graph (Example 5).

We can observe from the multiplier precedence graph of this structure (see Fig. 4.17(b)) that there are two inherent levels of serialism with respect to multipliers in this structure and that this level of serialism is independent of the number of cascaded sections.

Example 6 - Cascade (Avenhaus E) Structures

Using Circuit E(see [34]), we get the network configuration of Fig. 4.18(a). It was shown by Avenhaus that this class of structures is particularly suited for lowpass filter designs where the poles are close to z = 1. The system function can be expressed as

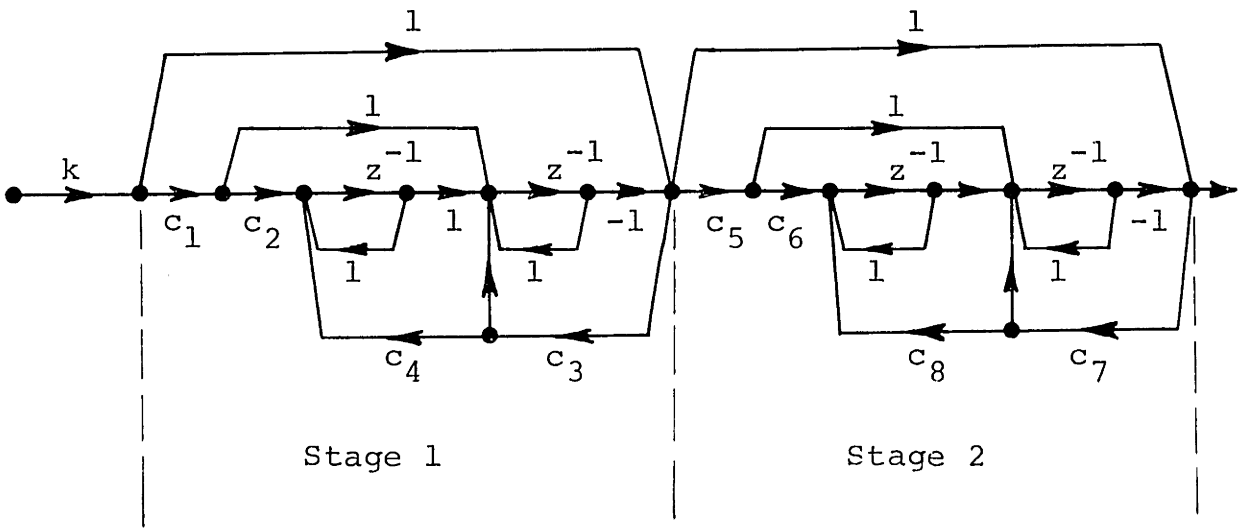
$$T(z) = k \cdot \frac{1 - (c_1+2)z^{-1} + (1+c_1-c_1c_2)z^{-2}}{1 + (c_3-2)z^{-1} + (1-c_3+c_3c_4)z^{-2}} \times$$

$$\frac{1 - (c_5+2)z^{-1} + (1+c_6-c_5c_6)z^{-2}}{1 + (c_7-2)z^{-1} + (1-c_7+c_7c_8)z^{-2}} .$$

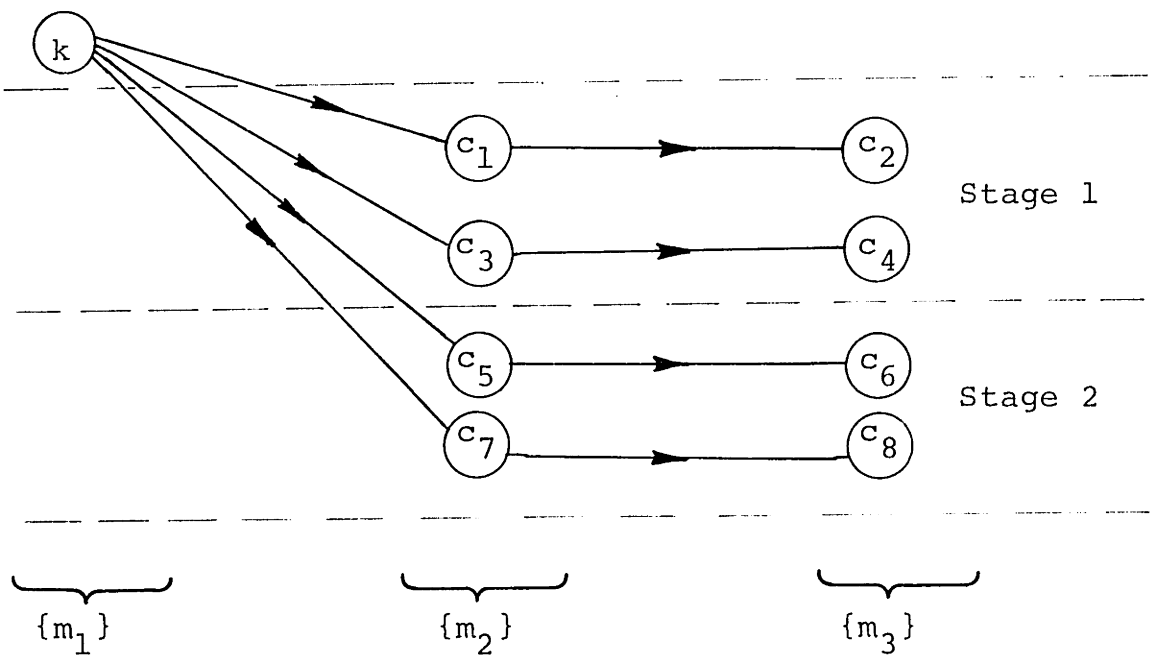
This class of structures has three levels of serialism in the multiplier precedence graph (see Fig. 4.18(b)). The number of levels is independent of the number of stages in the filter.

Example 7 - Cascade (Avenhaus F) Structures

A third form of second-order sections, Circuit F (see Avenhaus [34]), is particularly suited for the implementation of bandpass filters where the poles are located near the unit



(a)



(b)

Fig. 4.18 (a) Cascade (Avenhaus E) structure and
(b) its multiplier precedence graph (Example 6)

circle. This structure is illustrated in Fig. 4.19(a) and the system function can be expressed as

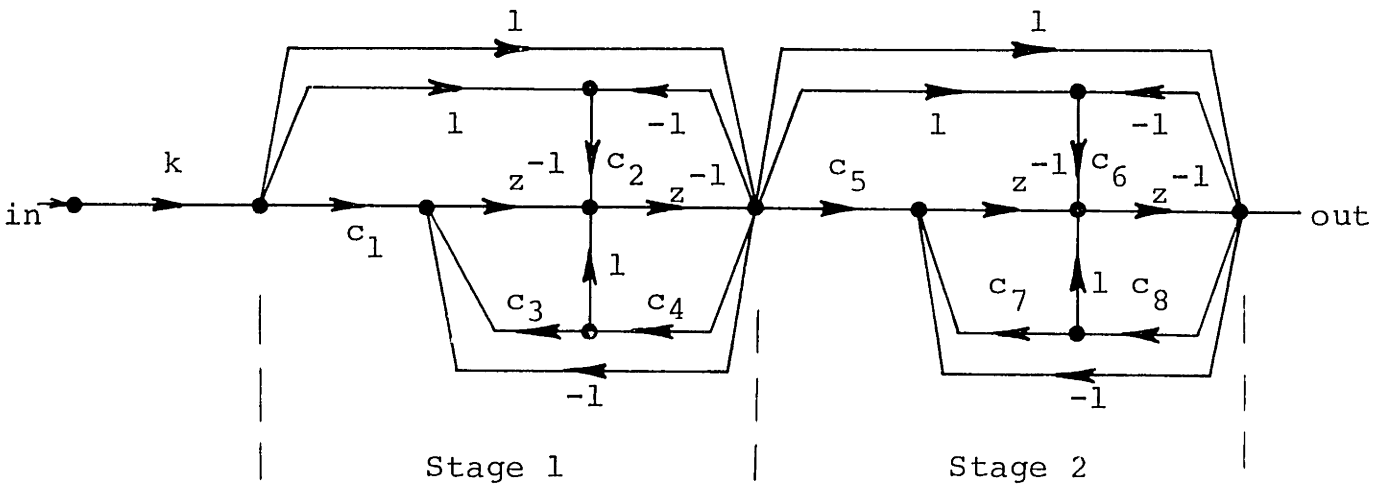
$$T(z) = k \cdot \frac{1 + c_2 z^{-1} + c_1 z^{-2}}{1 + (c_2 - c_4) z^{-1} + (1 - c_3 c_4) z^{-2}} \times$$

$$\frac{1 + c_6 z^{-1} + c_5 z^{-2}}{1 + (c_6 - c_8) + (1 - c_7 c_8) z^{-2}} .$$

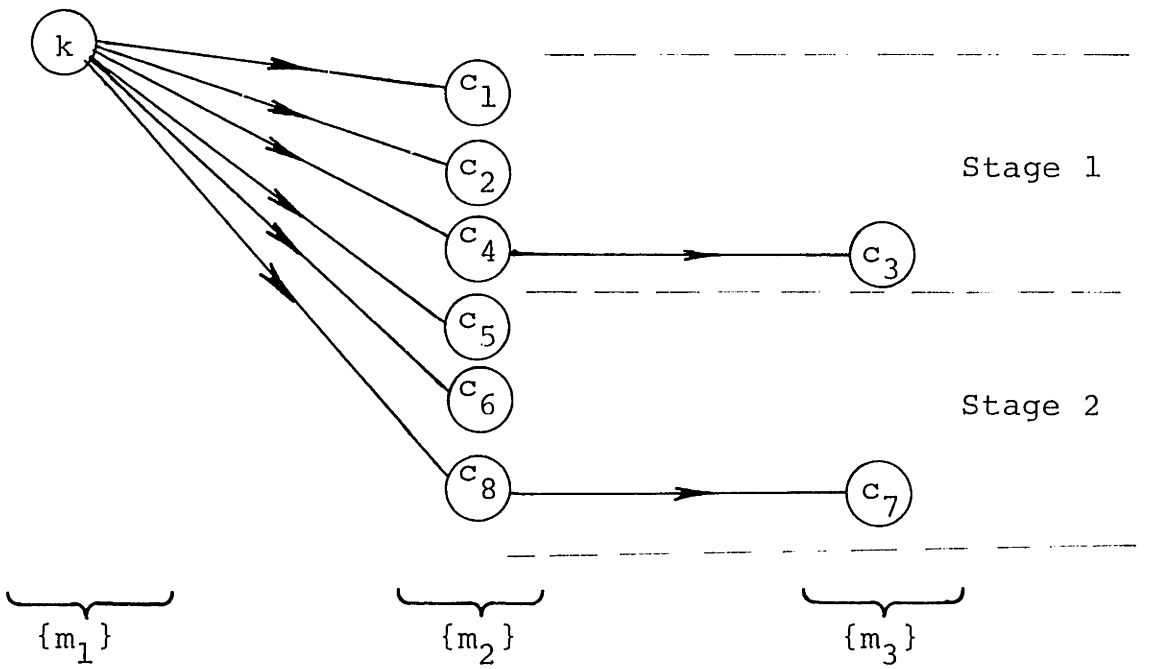
This class of structures also has three levels of serialism with respect to the multiplies (see Fig. 4.19(b)) and the number of levels is independent of the number of stages.

4.5.3 Continued Fraction Expansion Structures

A number of different classes of structures which can be derived by expressing the system function in terms of continued fraction expansions have been proposed by Mitra and Sherwood [31]. Three of these forms are examined in Examples 8, 9, and 10. A fourth form, type IIB, is noncomputable and will not be considered. The system functions for these forms can be expressed as:



(a)



(b)

Fig. 4.19 (a) Cascade (Avenhaus F) structure and
(b) its multiplier precedence graph (Example 7).

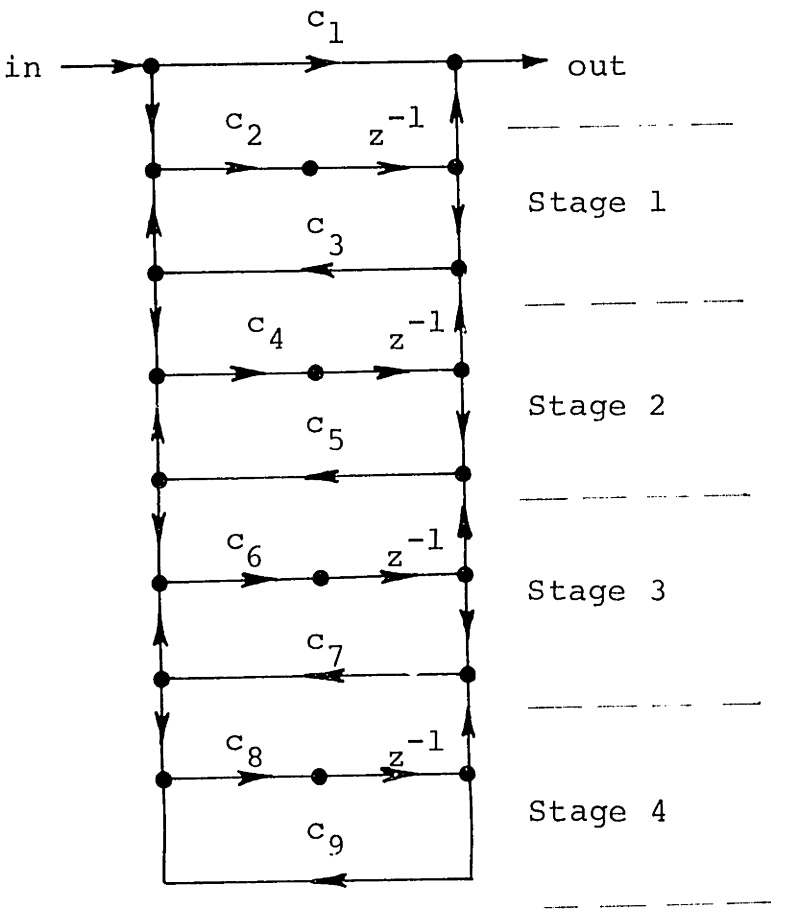
Example 8 - Continued Fraction Expansion IA (see Fig. 4.20)

$$T(z) = c_1 + \cfrac{1}{\left(\frac{1}{c_2}\right)z + \cfrac{1}{(-c_3) + \cfrac{1}{\left(\frac{1}{c_4}\right)z + \cfrac{1}{(-c_5) + \cfrac{1}{\left(\frac{1}{c_6}\right)z + \cfrac{1}{(-c_7) + \cfrac{1}{\left(\frac{1}{c_8}\right)z + \cfrac{1}{(-c_9)}}}}}}}$$

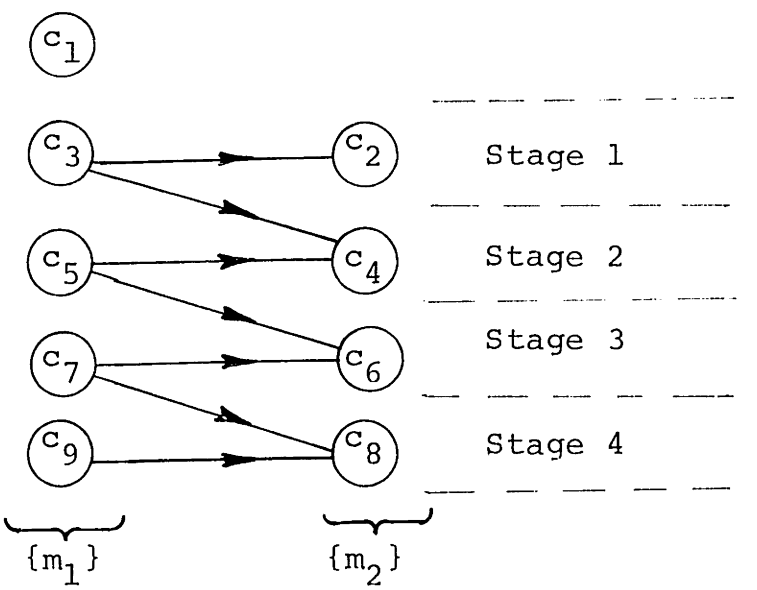
Example 9 - Continued Fraction Expansion IIA (see Fig. 4.21)

$$T(z) = c_1 + \cfrac{1}{\left(\frac{1}{c_2}\right)z - c_3 + \cfrac{1}{\left(\frac{1}{c_4}\right)z - c_5 + \cfrac{1}{\left(\frac{1}{c_6}\right)z - c_7 + \cfrac{1}{\left(\frac{1}{c_8}\right)z - c_9}}}}$$

and

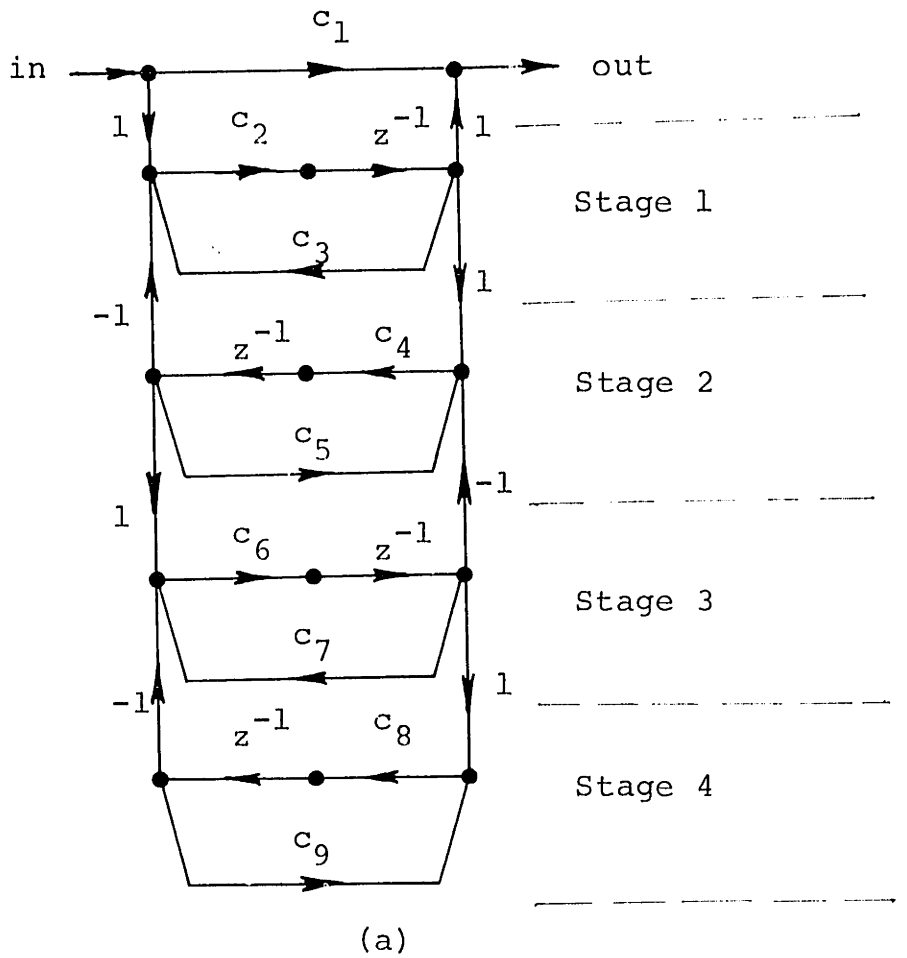


(a)

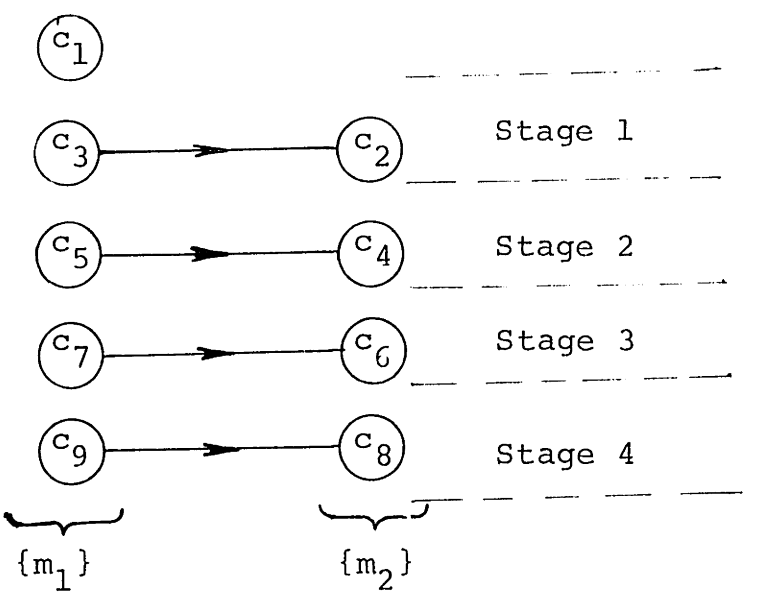


(b)

Fig. 4.20 (a) Continued fraction expansion structure, type 1A, and (b) its multiplier precedence graph (Example 8).



(a)



(b)

Fig. 4.21 (a) Continued fraction expansion structure, type IIA, and (b) its multiplier precedence graph (Example 9).

Example 10 - Continued Fraction Expansion IB (see Fig. 4.22)

$$T(z) = c_1 + \cfrac{1}{c_3 - c_2 z^{-1} + \cfrac{1}{c_5 - c_4 z^{-1} + \cfrac{1}{c_7 - c_6 z^{-1} + \cfrac{1}{c_9 - c_8 z^{-1}}}}}$$

In each of the above classes of continued fraction structures the inherent level of serialism with respect to multiplies is two and this level is independent of the order of the filter.

4.5.4 Ladder Structures

Another class of structures that has received considerable attention is the ladder structures. Various forms of ladder structures have been proposed by Mitra and Sherwood [18], Gray and Markel [35], and Fettweis [15], [16], [17], [36], [37]. The ladder structures of Fettweis will not be examined in this section as they apply more conveniently to system functions with zeros on the unit circle rather than to arbitrary system functions. A specific bandpass example for this class of structures will, however, be analyzed in Chapter 7.

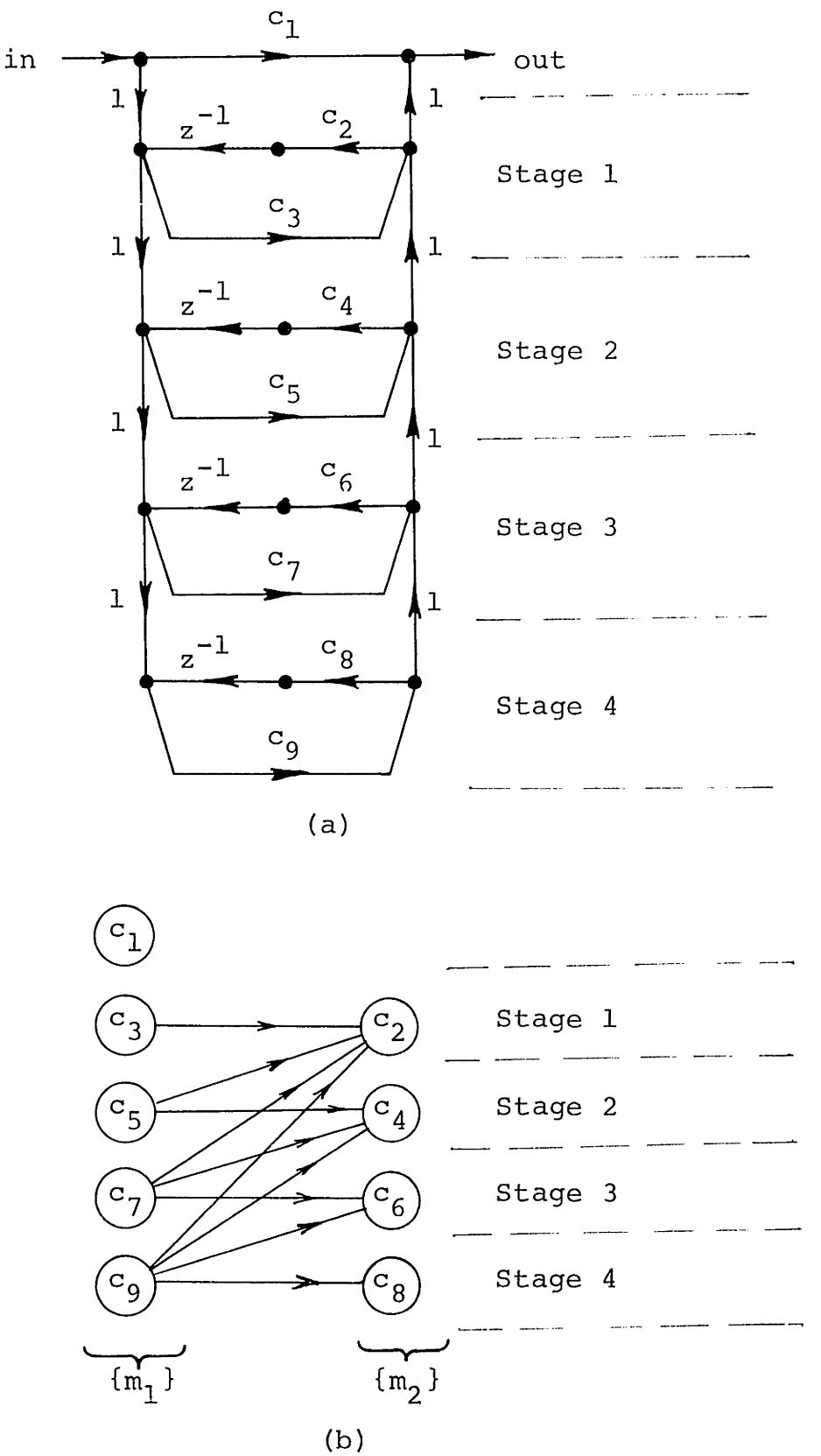


Fig. 4.22 (a) Continued fraction expansion structure, type IB, and (b) its multiplier precedence graph (Example 10).

Example 11 - Mitra and Sherwood Ladder Structures

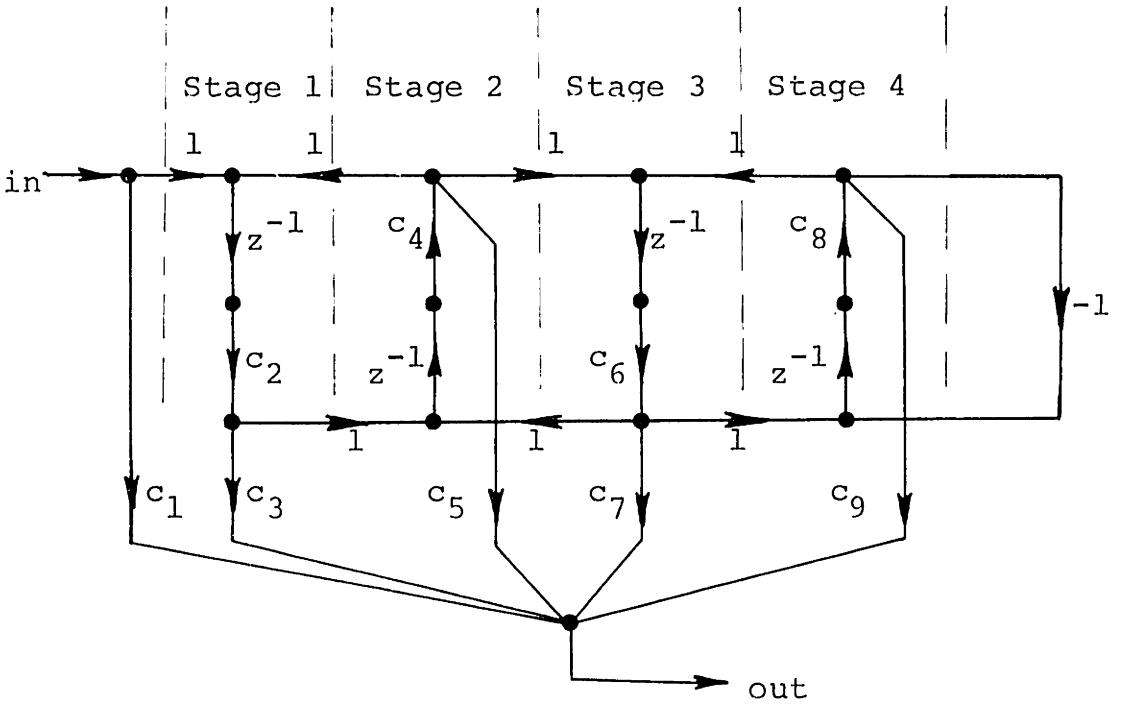
The implementation of a fourth-order system function by the class of ladder structures proposed by Mitra and Sherwood [18] results in the structure of Fig. 4.23. For this structure and other classes of ladder structures, expressions of the system function in terms of the coefficients in the structure are very complicated and not very very revealing. Consequently, such expressions will not be given. From the multiplier precedence graph of this structure (see Fig. 4.23 (b)), we see that there are two inherent levels of serialism with respect to multiplies and this level is independent of the order of the filter.

Example 12 - Gray and Markel Ladder Structures

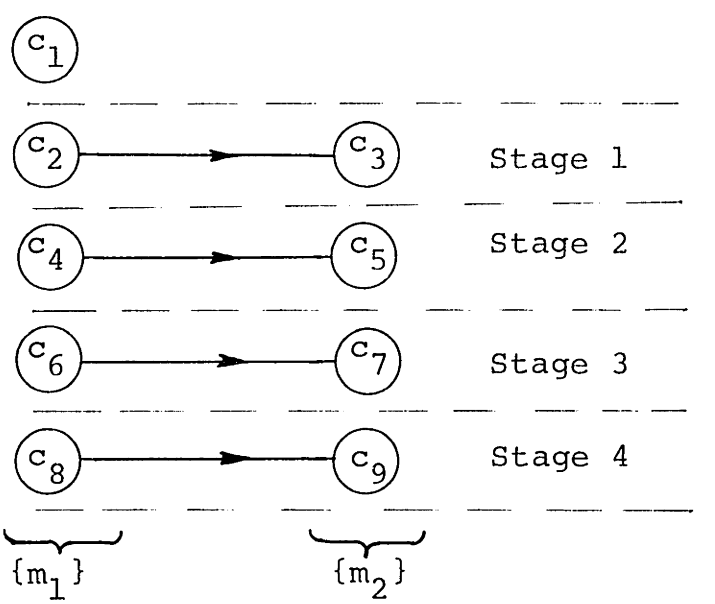
The ladder structure proposed by Gray and Markel [35] is illustrated in Fig. 4.24 for the fourth-order case. For this class of structures, we can see from the multiplier precedence graph that the level of serialism is 1 + n where n is the order of the filter. Consequently, the level of serialism increases linearly as the order of the system function. This class of structures is, therefore, highly serial in comparison to the other examples.

4.5.5 Comparison of Structures in Terms of Their t_m Versus M Graphs

To compare the various examples the t_m versus M graphs

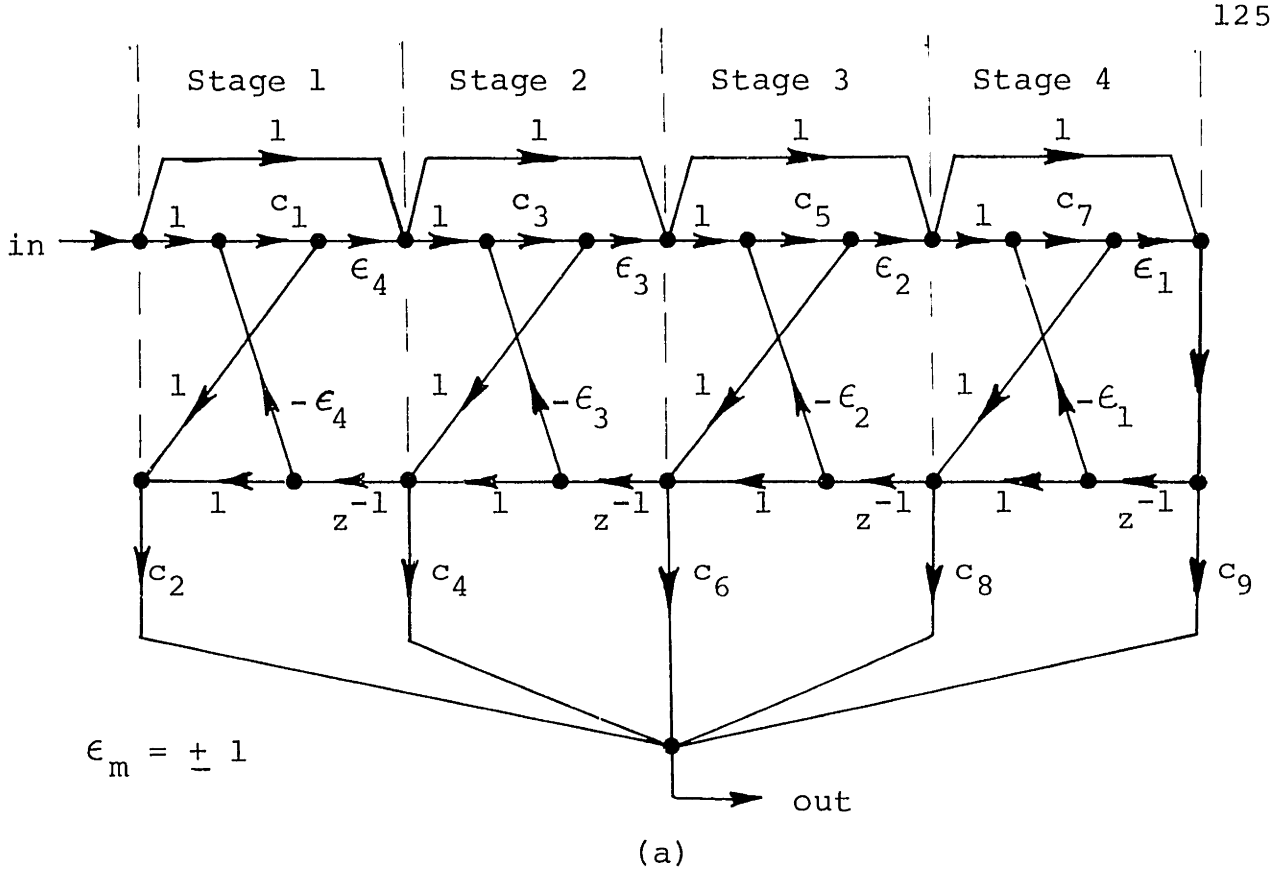


(a)

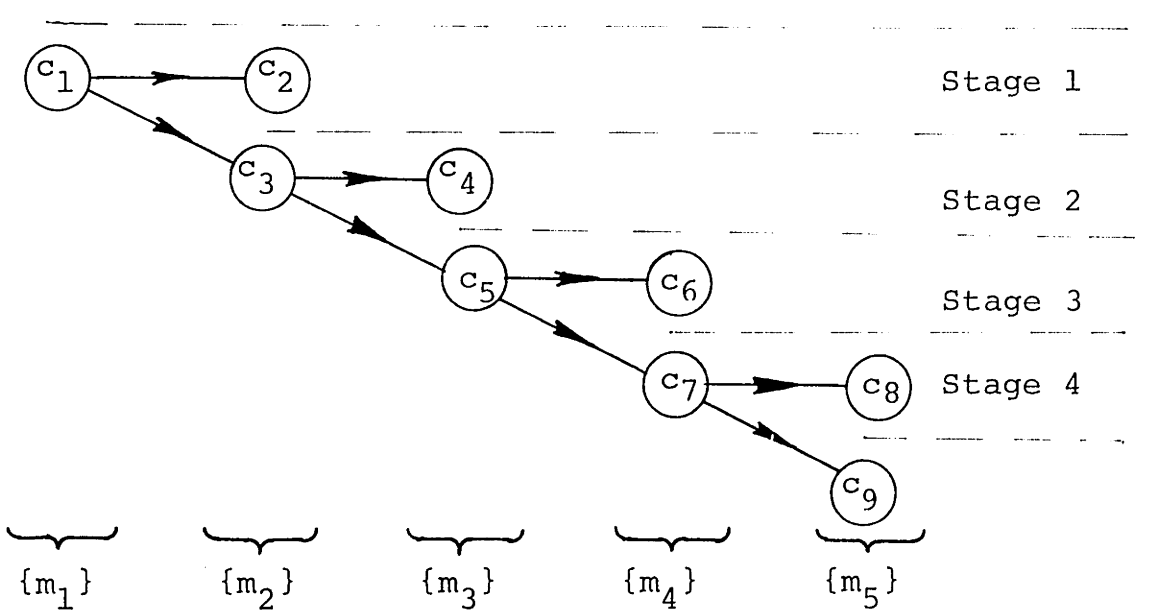


(b)

Fig. 4.23 (a) Ladder structure proposed by Mitra and Sherwood and (b) its multiplier precedence graph (Example 11).



(a)



(b)

Fig. 4.24 (a) Ladder structure proposed by Gray and Markel and (b) its multiplier precedence graph (Example 12).

of these structures are simultaneously plotted in Fig. 4.25. It is apparent from this plot that those structures which have greater levels of serialism (see Curves 3 and 4) or have more than the canonic number of multiplies (see Curve 5) require more computation time per filter cycle than the structures which are highly parallel, particularly in situations where we want to use more than one or two hardware multipliers in the implementation.

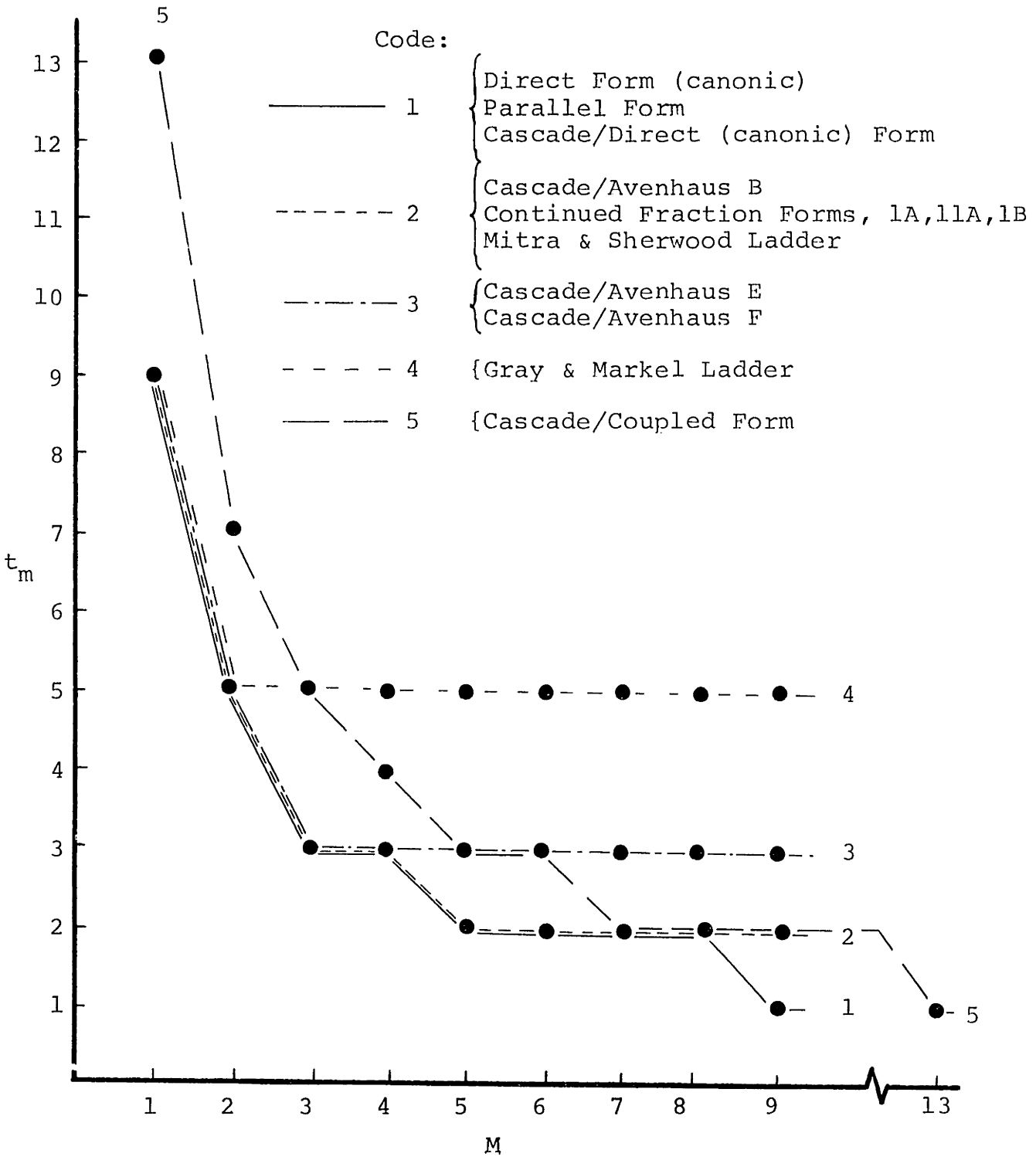


Fig. 4.25 Approximate minimum computation time, t_m, in multiplier cycles versus the number of multipliers, M, for Examples 1-12.

Chapter V

COMPUTER-AIDED NETWORK ANALYSIS OF DIGITAL FILTER STRUCTURES

5.1 INTRODUCTION

The objectives of this chapter are to show how the general matrix representation for digital networks and the network theory concepts and properties developed in Chapters 2 and 3 can be useful in the formulation of efficient computer-aided analysis algorithms for analyzing arbitrary digital filter structures. Some of these algorithms have been used in the development of a computer-aided digital network analysis package, CADNAP [38], which will be discussed briefly in Section 5.6 and in more detail in Appendix A.

The need for a general analysis tool such as CADNAP, which is capable of analyzing arbitrary network configurations, becomes apparent when we consider the large variety of possible structures which are candidates for filter implementations. To attempt to program a separate analysis routine for each possible structure which we might want to consider would be an enormous task and would be of limited use. With a general purpose interactive analysis package, this problem can easily be dealt with and any type of structure can be analyzed. The CADNAP program has been used extensively in the analysis of the examples in Chapters 6 and 7.

5.2 COMPUTATION OF THE IMPULSE RESPONSE

In Chapter 2, we saw that there is a one-to-one correspondence between an elementary digital network and its matrix representation. Consequently, by using the matrix approach we can conveniently duplicate or simulate all of the internal operations of the digital filter by performing matrix manipulations on the computer. To do this, we start with the matrix representation of the network

$$\underline{y}(k) = \underline{H}_c^t \underline{y}(k) + \underline{H}_d^t \underline{y}(k-1) + \underline{u}(k) \tag{5.1}$$

If we then compute the formulation similar to a state space form (see Chapter 2), we get

$$\underline{y}(k) = \underline{A}\underline{y}(k-1) + \underline{B}\underline{u}(k) \tag{5.2}$$

where

$$\underline{A} = (\underline{I} - \underline{H}_c^t)^{-1} \underline{H}_d^t \tag{5.3}$$

and

$$\underline{B} = (\underline{I} - \underline{H}_c^t)^{-1}. \tag{5.4}$$

To obtain the impulse response, $t_{ij}(k)$, from some input i to some output j we can assign the elements $\underline{u}(0)$ to be zero except for the i_{th} element which is set to 1. This corresponds to a unit sample at node i. The response at time k = 0 can then be computed as

$$\underline{y}(0) = \underline{\underline{B}}\underline{u}(0) \tag{5.5}$$

where $\underline{y}(-1)$, the initial conditions, have been assumed to be zero. For values of k, $k \geq 1$, we can assume that $\underline{u}(k) = 0$ and iterate the equation

$$\underline{y}(k) = \underline{\underline{A}}\underline{y}(k - 1) \tag{5.6}$$

to obtain the impulse response. The impulse response $t_{ij}(k)$ is obtained as $y_j(k)$. This procedure requires one matrix inversion to start and one matrix multiplication for each point in the impulse response. The size of the matrix is N x N, where N corresponds to the total number of nodes in the network.

An alternate method for computing the impulse response, if the network is computable, is to compute the N scalar equations in the matrix equation (5.1) directly, in consecutive order. For this approach, we must first arrange the matrix representation in a computable form. This can be achieved by the algorithm suggested in Theorem 2.2 of Chapter 2. To obtain the impulse response $t_{ij}(k)$, we again assume zero initial conditions for $\underline{y}(k- 1)$, place a unit sample in node i at time k = 0, and iterate the matrix equation (5.1). This approach is also useful for simulating the nonlinear effects of the digital filter such as limit cycles, because the equations which we are iterating are exactly those which

would be implemented in the filter. This can be achieved by simulating on the computer the finite precision arithmetic actually used in the implementation of the filter.

5.3 COMPUTATION OF THE FREQUENCY RESPONSE

Another analysis feature which we often want in a computer-aided analysis package is the ability to compute frequency responses. For this, we have a variety of methods at our disposal. Again, for general analysis of arbitrary structures, we begin with the matrix representation

$$\underline{Y}(z) = \underline{H}_c^t \underline{Y}(z) + \underline{H}_d^t \underline{Y}(z)\ z^{-1} + \underline{U}(z). \tag{5.7}$$

One approach for computing the frequency response is to perform the matrix inversion in the relation

$$\underline{T}(z) = (\underline{I} - \underline{H}_c^t - \underline{H}_d^t\ z^{-1})^{-1}. \tag{5.8}$$

This must be done at each frequency z. To obtain the system function $T_{ij}(z)$, we then pick out the appropriate element from $\underline{T}(z)$. In general, this approach requires an excessive amount of computation and is prone to numerical errors due to round-off in the computations [8]. Consequently, it is not a very desirable method to use.

A more appropriate way of computing the frequency response is to write the matrix equation in the form

$$(\underline{I} - \underline{H}_c^t - \underline{H}_d^t z^{-1})\ \underline{Y}(z) = \underline{U}(z). \tag{5.9}$$

As the matrices $\underline{I}$, $\underline{H}_c^t$, and $\underline{H}_d^t$ are known and the input $\underline{U}(z)$ is set to zero, except for a unit sample at the appropriate input i, the only variable to be determined is $\underline{Y}(z)$. Therefore, it is easy to see that equation (5.9) is in the form of N linear simultaneous equations in N unknowns. These equations can easily be solved by computer using complex arithmetic with an algorithm such as Gaussian elimination [8], [39], [40]. This approach requires the solution of N equations in N unknowns for each frequency desired. The frequency response $T_{ij}(z)$ for a specific input i and a specific output j is obtained as the j^{th} element of $\underline{Y}(z)$, where $\underline{U}(z)$ is appropriately defined above.

A special case of interest for computing the frequency response is when the network is computable and nonrecursive. For this case, we can express the matrix representation in feedback-free form by applying the algorithm suggested in the proof of Theorem 2.3 (see Chapter 2). We can then proceed to solve the equations in (5.7) consecutively for $Y_i(z)$ for $i = 1, 2, \ldots, N$. In effect, we are still solving N linear simultaneous equations by this method but, because of the special nature of the network, the matrix representation can be expressed in a convenient manner for doing this.

In the above analysis procedures of solving N equations in N unknowns, it is necessary to resort to using complex arithmetic. For computer solution, it is sometimes more convenient to solve sets of linear equations for real arithmetic.

For example, in the APL language the solving of real simultaneous linear equations is a primative operation. For such cases, the equations can be transposed into a form such that the real and imaginary parts of $\underline{Y}(z)$ are evaluated separately by solving two sets of N simultaneous linear equations using real arithmetic. To show this, we will consider the case where $z = e^{j\omega}$. We start with the form

$$\underline{Y}(e^{j\omega}) = e^{-j\omega}\underline{\underline{A}}\underline{Y}(e^{j\omega}) + \underline{\underline{B}}\underline{U}(e^{j\omega}), \tag{5.10}$$

where $\underline{\underline{A}}$ and $\underline{\underline{B}}$ are defined in (5.3) and (5.4), respectively. We can rewrite (5.10) in the form

$$(\underline{I} - \underline{\underline{A}} \cos \omega + j\underline{\underline{A}} \sin \omega)\ \underline{Y}(e^{j\omega}) = \underline{\underline{B}}\underline{U}(e^{j\omega}). \tag{5.11}$$

By premultiplying by $\underline{I} - \underline{\underline{A}} \cos \omega - j\underline{\underline{A}} \sin \omega$ and cancelling terms, we get

$$(\underline{I} - 2\underline{\underline{A}} \cos \omega + \underline{\underline{A}}^2)\ \underline{Y}(e^{j\omega}) = (\underline{I} - \underline{\underline{A}} \cos \omega - j\underline{\underline{A}} \sin \omega) \cdot \underline{\underline{B}}\underline{U}\ (e^{j\omega}). \tag{5.12}$$

In the evaluation of a frequency response $T_{ij}(e^{j\omega})$, we set $\underline{U}(e^{j\omega})$ to be zero except for element i which is 1. Therefore, $\underline{\underline{B}}\underline{U}(e^{j\omega})$ is real and (5.12) can be separated into its real and imaginary parts as

$$\underline{\underline{C}}\underline{Y}_R(e^{j\omega}) = (\underline{I} - \underline{\underline{A}} \cos \omega)\ \underline{\underline{B}}\underline{U}\ (e^{j\omega}) \tag{5.13}$$

and

$$\underline{C}\underline{Y}_I(e^{j\omega}) = (-\underline{A} \sin \omega)\ \underline{B}\underline{U}(e^{j\omega}) \tag{5.14}$$

where

$$\underline{C} = \underline{I} - 2\underline{A} \cos \omega + \underline{A}^2, \tag{5.15}$$

$$\underline{Y}_R(e^{j\omega}) = \text{the real part of } \underline{Y}(e^{j\omega}),$$

and

$$\underline{Y}_I(e^{j\omega}) = \text{the imaginary part of } \underline{Y}(e^{j\omega}).$$

The solution is then obtained by recognizing that (5.13) and (5.14) are each in the form of N simultaneous linear equations in real variables which can be solved using real arithmetic. The transfer function $T_{ij}(e^{j\omega})$ is obtained as the j^{th} element of $\underline{Y}_R(e^{j\omega})$ for the real part and the j^{th} element of $\underline{Y}_I(e^{j\omega})$ for the imaginary part. In addition, in the formulation of equations (5.13) and (5.14), it is necessary to compute an N x N matrix inversion (using real arithmetic) to obtain $\underline{A}$ and $\underline{B}$ in (5.3) and (5.4) and also, to perform a number of matrix multiplies. While this can correspond to a considerable amount of computation, if we are computing the frequency response at many frequencies (as is usually the case), it is only necessary to compute this matrix inversion once to establish the equations in the form of (5.13) and (5.14). After this, we can compute the frequency response

using real arithmetic by only solving two sets of N equations in N unknowns for each frequency. Therefore, aside from the one initial matrix inversion, this method is as efficient computationally as solving the original linear equations with complex arithmetic.

While frequency analysis by solving linear equations is considerably more efficient computationally than matrix inversion, if we are interested in computing the frequency response for a relatively large number of frequencies, the process of solving N linear simultaneous equations at each frequency may still require a great deal of computation. One way to improve upon the efficiency of these computations is to attempt to exploit the sparsity of the network equations in the process of solving the linear simultaneous equations. The matrix representation of a digital network (5.7) will generally contain a large percentage of zero elements, perhaps on the order of 75% or more. Various techniques have been recently proposed [41], [42] for reducing the amount of computation necessary to solve linear simultaneous equations of this type. Although these methods have been developed for computer-aided analysis of conventional analog circuits, many of them apply equally well for the case of digital networks once an appropriate matrix representation, such as (5.7), is defined.

An alternate approach to the problem of computing the frequency response of a network, when many frequency points

are desired, is to first perform the more difficult task of computing the poles and zeros of the system function. Once these poles and zeros are known, it is an easy matter to compute the frequency response very efficiently for a large number of frequencies by simply evaluating the ratio of factored polynomials comprising the system function. The problem can be formulated by the state space representation of the network equations

$$\underline{Y}(z) = \underline{A}\underline{Y}(z)z^{-1} + \underline{B}\underline{U}(z), \tag{5.16}$$

where the transfer function can be expressed as

$$\underline{T}^{t}(z) = z(\underline{I}z - \underline{A})^{-1}\,\underline{B}. \tag{5.17}$$

Using the definition of a matrix inverse, we can express $\underline{T}^{t}(z)$ as

$$\underline{T}^{t}(z) = \frac{z\ \mathrm{adj}[\underline{I}z - \underline{A}]\underline{B}}{\det[\underline{I}z - \underline{A}]}$$

$$= \frac{\underline{P}(z)}{Q(z)}\ . \tag{5.18}$$

We can note from this relation that the poles of a particular transfer function $T_{ij}(z)$ correspond to the roots of the polynomial $Q(z)$, which do not cancel with the zeros of $P_{ji}(z)$, and the zeros of $T_{ij}(z)$ are the noncancelling roots of $P_{ji}(z)$. We can also note from state space theory or linear algebra

that the roots of Q(z) are exactly the eigenvalues of the matrix $\underline{A}$. Therefore, the problem of determining the poles reduces to the problem of computing the eigenvalues of a matrix. A well-known and, perhaps, the most accurate of known methods for accomplishing this task, is the Q-R algorithm [43], [44]. The problem of determining the zeros of $P_{ji}(z)$ is more difficult, but it can also be formulated into a problem of determining the eigenvalues of a matrix. This has been treated in detail for the case of continuous time state space representations by Brockett [45] and by Sandberg and So [46]. It should also apply, with slight modification, to the state space representation given above. One difficulty that is encountered in attempting to evaluate the poles and zeros of the system function from the matrix representation is that if there are more nodes in the network than poles or zeros in the system function, then there will be roots of $P_{ij}(z)$ and Q(z) which will cancel. Because of numerical roundoff in the computations of these roots, we are faced with the problem of deciding which poles and zeros will cancel and which will not. Furthermore, if the number of nodes in the network is very large compared to the number of poles and zeros in the system function, the matrix will be relatively large and the problem can become numerically sensitive and prone to error. Thus, while the method of first computing the poles and zeros of the system function is an attractive scheme for efficiently evaluating the frequency response of a

network for a large number of frequencies, it must be used with caution.

5.4 COMPUTATIONAL METHODS FOR SENSITIVITY ANALYSIS OF DIGITAL FILTERS

5.4.1 First-Order Sensitivity Definitions

The multiparameter sensitivity of a transfer function or system function, $T = T(z)$, of a digital filter with respect to its coefficients c_i $(i = 1, \ldots, n)$ can be defined in several ways. Three basic definitions which have classically been used in filter theory [7], [27], [28], [29], [30], [47], [48], [49], [50] are considered here and the general relationships among them are given.

The first definition will be referred to as the "basic sensitivity" or simply the "sensitivity" of the filter. It is the vector of partial derivatives (gradient) of the transfer function T with respect to its coefficients,

$$\underline{S}[T] \triangleq \nabla T = \left(\frac{\partial T}{\partial c_1}, \ldots \frac{\partial T}{\partial c_n}\right)^t, \tag{5.19}$$

where

$$T \triangleq T(c_1, c_2, \ldots c_n).$$

A single element of this vector can be expressed as

$$S_i[T] = \lim_{\Delta c_i \to 0} \frac{\Delta T}{\Delta c_i} = \frac{\partial T}{\partial c_i} . \tag{5.20}$$

The first-order (incremental) variation of T, ΔT, with respect to incremental changes in the coefficients Δc_i can be determined via the sensitivities as

$$\Delta T \approx \sum_{i=1}^{n} S_i[T] \, \Delta c_i = \underline{S}^t[T] \, \Delta \underline{c}, \tag{5.21}$$

where

$\underline{S}^t[T]$ = the transpose of the sensitivity vector

$\Delta \underline{c}$ = the vector of incremental changes in the coefficients

$= (\Delta c_1, \Delta c_2, \ldots, \Delta c_n)^t$.

A second definition that has been widely used in analog filter theory is referred to as the "relative sensitivity". A single element of the relative sensitivity vector is defined as

$$S_i^r[T] \triangleq \lim_{\Delta c_i \to 0} \frac{\Delta T/T}{\Delta c_i/c_i} = \frac{c_i}{T} \frac{\partial T}{\partial c_i}$$

$$\triangleq \frac{\partial \ln T}{\partial \ln c_i} , \tag{5.22}$$

and the sensitivity vector is then given as

$$\underline{S}^{r}[T] \triangleq \left(\frac{\partial \ln T}{\partial \ln c_1}, \frac{\partial \ln T}{\partial \ln c_2}, \ldots \frac{\partial \ln T}{\partial \ln c_n}\right)^{t}. \quad (5.23)$$

The reason for the popularity of this definition is that it is more closely associated with percentage changes of the system function with respect to percentage changes in the coefficients (element values). It has been particularly useful in tolerance studies. In the case of digital filters, this definition would seem to be most appropriate for filters with floating-point coefficients.

We see from (5.22) that the relative sensitivities can be expressed in terms of the sensitivities as

$$S_i^r[T] = \frac{c_1}{T} S_i[T]. \quad (5.24)$$

The relative (incremental) change of the system function with respect to relative (incremental) changes in the coefficients can be expressed as

$$\Delta T^r \triangleq \frac{\Delta T}{T} \approx \underline{S}^{r^t}[T]\ \underline{\Delta c}^r, \quad (5.25)$$

where

$$\Delta c_1^r = \frac{\Delta c_1}{c_1} \quad (5.26)$$

= the relative change in the coefficients.

A third definition of sensitivity which is often

encountered is the "semirelative sensitivity". This definition can be expressed as

$$S_i^s[T] \triangleq \lim_{\Delta c_1 \to 0} \frac{\Delta T/T}{\Delta c_1} = \frac{1}{T}\frac{\partial T}{\partial c_1} \triangleq \frac{\partial \ln T}{\partial c_1} \tag{5.27}$$

or

$$S_i^s[T] = \frac{1}{T} S_i[T] \tag{5.28}$$

and

$$\underline{S}^s = \nabla(\ln T) = \left(\frac{\partial \ln T}{\partial c_1}, \frac{\partial \ln T}{\partial c_2}, \ldots \frac{\partial \ln T}{\partial c_n}\right)^t. \tag{5.29}$$

This definition seems more appropriate for digital filters with fixed-point arithmetic since it relates relative changes in the system function in terms of absolute changes in the coefficients. This relation can be expressed as

$$\Delta T^r \triangleq \frac{\Delta T}{T} \approx \underline{S}^{s^t}[T]\ \underline{\Delta c}. \tag{5.30}$$

Often, in the case of filters, it is desired to have the sensitivity of the magnitude of the system function $|T|$. These sensitivities can be derived quite easily from the basic sensitivities and are given below:

$$S_i[|T|] = \frac{1}{|T|} \mathrm{Re}(T^* S_i[T]) = \mathrm{Re}\left(\frac{|T|}{T} S_i[T]\right), \tag{5.31}$$

$$S_i^r[|T|] = \mathrm{Re}\left(\frac{c_i}{T} S_i[T]\right), \tag{5.32}$$

$$S_i^s[|T|] = \mathrm{Re}\left(\frac{1}{T} S_i[T]\right). \tag{5.33}$$

It has been assumed in the derivations of the above expressions that the coefficients c_i are real.

5.4.2 Efficient Computation of Network Sensitivities

From equations (5.24), (5.28), (5.31), (5.32), and (5.33), it can be seen that all of the first-order sensitivity functions of interest can be easily computed once the basic sensitivities, as defined in (5.19) and (5.20), are known. In this section, a method will be described by which all of the basic sensitivities of a digital network at a given frequency can be determined exactly with only two complete analyses of the network at that frequency [27].

We begin with the matrix representation (5.7) and assume that the transfer function of interest is $T_{k\ell} = T_{k\ell}(z)$. In Chapter 3, we derived a first-order sensitivity expression which relates the sensitivity of $T_{k\ell}$, with respect to a branch coefficient of a branch from some node p to some node q, in terms of the product of two transfer functions in the network. For a branch with a simple coefficient c, this expression is

$$S_c[T_{k\ell}] = \frac{\partial T_{k\ell}}{\partial c} = T_{kp}T_{q\ell} \tag{5.34}$$

and for a branch with a coefficient d and a delay, it is

$$S_d[T_{k\ell}] = \frac{\partial T_{k\ell}}{\partial d} = z^{-1}T_{kp}T_{q\ell}. \tag{5.35}$$

Therefore, in general, if we know all of the transfer functions from the input node k to the nodes p (p = 1, ..., N), where N is the total number of nodes in the network and all of the transfer functions from the nodes q (q = 1, ..., N) to the output node ℓ, then we have sufficient information to compute all of the first-order sensitivities $\underline{S}[T_{k\ell}]$ for the network function $T_{k\ell}$.

The first set of transfer functions T_{kp} can be obtained by solving the following set of simultaneous linear equations

$$(\underline{I} - \underline{H}_c^t - \underline{H}_d^t z^{-1})\ \underline{Y}(z) = \underline{U}(z). \tag{5.36}$$

The elements of the vector of input signals $\underline{U}(z)$ are all set to zero except for the one that corresponds to node k. This element is set to unity (the z transform of a unit sample). Assuming that $\underline{H}_d$, $\underline{H}_c$, and z are known, equation (5.36) can be solved for $\underline{Y}(z)$ by a procedure such as Gaussian Elimination (using complex arithmetic). The solution $\underline{Y}(z)$ represents the appropriate transfer functions from the input node

k to all of the nodes in the network, that is,

$$T_{kp}(z) = Y_p(z) \tag{5.37}$$

The second set of transfer functions $T_{q\ell}(z)$ can be obtained in a similar manner by analyzing the transpose network. In the matrix formulation (5.7) this corresponds to taking the transpose of the matrices $\underline{H}_c$ and $\underline{H}_d$. It has been shown that the transpose network is interreciprocal with the original network (see Chapter 3), that is,

$$T_{q\ell}(z)\Big|_{\substack{\text{original}\\\text{network}}} = T_{\ell q}(z)\Big|_{\substack{\text{transpose}\\\text{network}}} \tag{5.38}$$

for all q (q = 1, ..., N) and ℓ (ℓ = 1, ..., N). Therefore, by solving the transpose network with an input at node ℓ, we can obtain all of the transfer functions $T_{q\ell}(z)$ of the original network (q = 1, ..., N). This corresponds to solving the set of simultaneous linear equations

$$(\underline{I} - \underline{H}_c - \underline{H}_d z^{-1})\, \underline{Y}(z) = \underline{U}(z) \tag{5.39}$$

for $\underline{Y}(z)$ with all of the elements of $\underline{U}(z)$ equal to zero except for the one which corresponds to node ℓ, which is set to unity. The desired transfer functions $T_{q\ell}(z)$ can then be obtained with the aid of (5.38) from the solution of (5.39) by the relation

$$T_{q\ell}(z) = Y_q(z). \tag{5.40}$$

Thus, from the solution of two sets of simultaneous linear equations, (5.36) and (5.39), and with the aid of (5.34), (5.35), (5.37), and (5.40), all of the basic sensitivities of the system function $T_{k\ell}(z)$ can be determined. This technique of computing network sensitivities is similar, in some respects, to the adjoint network techniques used by Director and Rohrer [51] in the computation of element sensitivities in analog circuits.

In general, the sets of linear equations (5.36) and (5.39) must be solved using complex arithmetic. In a manner similar to the case of the frequency response analysis, these equations can be transposed to a form in which the real and imaginary components of $\underline{Y}(z)$ can be solved separately, each requiring the solution of a set of simultaneous linear equations in real numbers. To do this, consider for the sake of convenience the case for $z = e^{j\omega}$. Then, (5.36) becomes

$$(\underline{I} - \underline{H}_c^t - \underline{H}_d^t (\cos \omega - j \sin \omega))\ \underline{Y}(e^{j\omega}) = \underline{U}(e^{j\omega}). \tag{5.41}$$

Premultiplying (5.41) by $\underline{I} - \underline{H}_d^t(\underline{I} - \underline{H}_c^t)^{-1}(\cos \omega + j \sin \omega)$ gives

$$\underline{D}(\underline{Y}_R(e^{j\omega}) + j\underline{Y}_I(e^{j\omega})) = (\underline{I}-\underline{H}_d^t(\underline{I}-\underline{H}_c^t)^{-1}\ (\cos\omega+j\ \sin\omega))\underline{U}(e^{j\omega}), \tag{5.42}$$

where

$$\underline{Y}_R(e^{j\omega}) = \text{the real part of } \underline{Y}(e^{j\omega}),$$

$$\underline{Y}_I(e^{j\omega}) = \text{the imaginary part of } \underline{Y}(e^{j\omega}),$$

and

$$\underline{D} = \underline{I} - \underline{H}_c^t + \underline{H}_d^t(I-\underline{H}_c^t)^{-1}\ \underline{H}_d^t - 2\underline{H}_d^t \cos\ \omega. \tag{5.43}$$

For a unit sample response $\underline{U}(e^{j\omega}) = \underline{U}_R$ is real and the real and imaginary parts of (5.42) can be separated as

$$\underline{DY}_R(e^{j\omega}) = \underline{U}_R - \underline{H}_d^t(I-\underline{H}_c^t)^{-1}\ \underline{U}_R \cos\ \omega \tag{5.44}$$

and

$$\underline{DY}_I(e^{j\omega}) = -\underline{H}_d^t(\underline{I} - \underline{H}_c^t)^{-1}\ \underline{U}_R \sin\ \omega. \tag{5.45}$$

Equations (5.44) and (5.45) can be solved for $\underline{Y}_R(e^{j\omega})$ and $\underline{Y}_I(e^{j\omega})$ as linear sets of simultaneous equations in real variables. The matrix inversions in (5.43), (5.44), and (5.45) also require only real variables. It may seem, at first galnce, that this approach requires much more computation since it is necessary to perform a matrix inversion and several extra matrix multiplications. However, when we are interested in computing the sensitivities for a large set of frequencies (as is usually the case), the matrix inversions and matrix multiplications all have to be computed only once

for the entire set of frequencies. Thus, the solutions to (5.44) and (5.45) can be computed quite efficiently. In a similar manner, the solution to (5.39) can be obtained by solving two sets of real simultaneous linear euqations which are related to (5.44) and (5.45) by the transposes of $\underline{D}$, $\underline{H}_c$, and $\underline{H}_d$ as

$$\underline{D}^t \underline{Y}_R(e^{j\omega}) = \underline{U}_R - \underline{H}_d(\underline{I} - \underline{H}_c)^{-1} \underline{U}_R \cos \omega \qquad (5.46)$$

and

$$\underline{D}^t \underline{Y}_I(e^{j\omega}) = -\underline{H}_d(\underline{I} - \underline{H}_c)^{-1} \underline{U}_R \sin \omega. \qquad (5.47)$$

For the case of nonrecursive structures the unit sample responses of the network are all of finite duration. In this case, the simultaneous equations (5.36) and (5.39) can be solved in the time domain by placing a unit sample in the appropriate input of the network and the transpose network and iterating them to obtain the appropriate unit sample responses. Then, $T_{kp}(z)$ and $T_{q\ell}(z)$ can be obtained by taking the z transforms of these unit sample responses. For special classes of frequency points these transforms can be performed very efficiently with the aid of the FFT algorithm, frequency warping techniques [52], and the chirp z transform algorithm [2], [3].

5.5 EFFICIENT METHODS FOR OTHER TYPES OF ANALYSIS OF ARBITRARY DIGITAL NETWORKS

In addition to the analysis of the impulse response, frequency response, and network sensitivities, there are many other types of analyses that we may want to perform on a digital filter. In this section, we will briefly point out some of these types of analyses and discuss ways of efficiently performing them in a general computer-aided analysis context.

One important parameter which we may be interested in is the group delay, τ, of a filter system function T. A useful and efficient method for computing the group delay [27], [53] is to resort to the relationship developed in Chapter 3 which expresses the group delay in terms of specific coefficient sensitivities in the structure. This expression is given as

$$\tau = \sum_{i=1}^{m} \text{Re } S_i^r[T], \tag{5.48}$$

where the relative sensitivities $S_i^r[T]$ are taken with respect to the coefficients in all of the coefficient and delay branches in the network, that is, all of the nonzero elements of $\underline{H}_d$. Alternatively, as the relative sensitivities of zero valued coefficients are zero, we could take the sum in (5.48) to extend over <u>all</u> elements in $\underline{H}_d$ whether they are zero or nonzero. We know from the previous section that the sensitivities $S_i^r[T]$, at a particular frequency, can all be computed simultaneously with only two network analyses at that frequency.

Consequently, using (5.48) we can also compute the group delay exactly with only two network analyses.

In a similar manner to the computation of the group delay, we can compute the derivative of the log magnitude of the system function with respect to the frequency. To do this, we begin with the relation (see Chapter 3)

$$\frac{\partial \ln |T|}{\partial \omega} = \sum_{i=1}^{m} \text{Im } S_i^r[T]. \tag{5.49}$$

In this case, we sum the imaginary parts of the relative sensitivities with respect to the coefficients in $\underline{H}_d$. Again, this can be accomplished with only two network analyses per frequency point. If we want the derivative of $|T|$ with respect to ω, we can write

$$\frac{\partial \ln |T|}{\partial \omega} = \frac{1}{|T|} \frac{\partial |T|}{\partial \omega} .$$

Applying this to (5.49) and multiplying by $|T|$ gives the desired relation

$$\frac{\partial |T|}{\partial \omega} = \frac{1}{|T|} \sum_{i=1}^{m} \text{Im } S_i^r[T]. \tag{5.50}$$

The calculation of $|T|$ in (5.50) does not require any extra computation as we can obtain it from either of the two analyses used to obtain the coefficient sensitivities.

5.6 AN OUTLINE OF THE COMPUTER-AIDED DIGITAL NETWORK ANALYSIS PACKAGE (CADNAP)

5.6.1 General Features of CADNAP

CADNAP is a set of general purpose programs for manipulating and analyzing digital networks. It is capable of handling digital structures of any arbitrary configuration and is designed to be highly interactive with the user [38]. In its present state of development, CADNAP contains operators to analyze network response functions from any point in a network to any other point. It can compute unit sample responses, frequency responses, and coefficient sensitivities of any transfer function in a network. Coefficients can be rounded or truncated in fixed-point arithmetic and structural manipulations, such as taking the transpose of a network, can be performed.

The basic functional approach used by CADNAP is depicted in Fig. 5.1. The common parameter in all of the analysis and editing operators is a network. A network is stored in compact form, as a vector, on the computer. The network is reconstructed in terms of its matrix representation from this vector, only when it is needed. In this way, several networks can be stored simultaneously without consuming excessive amounts of storage. The basic operators in CADNAP can be classified into two types: editing operators and analysis operators. The editing operators take a network, modify it

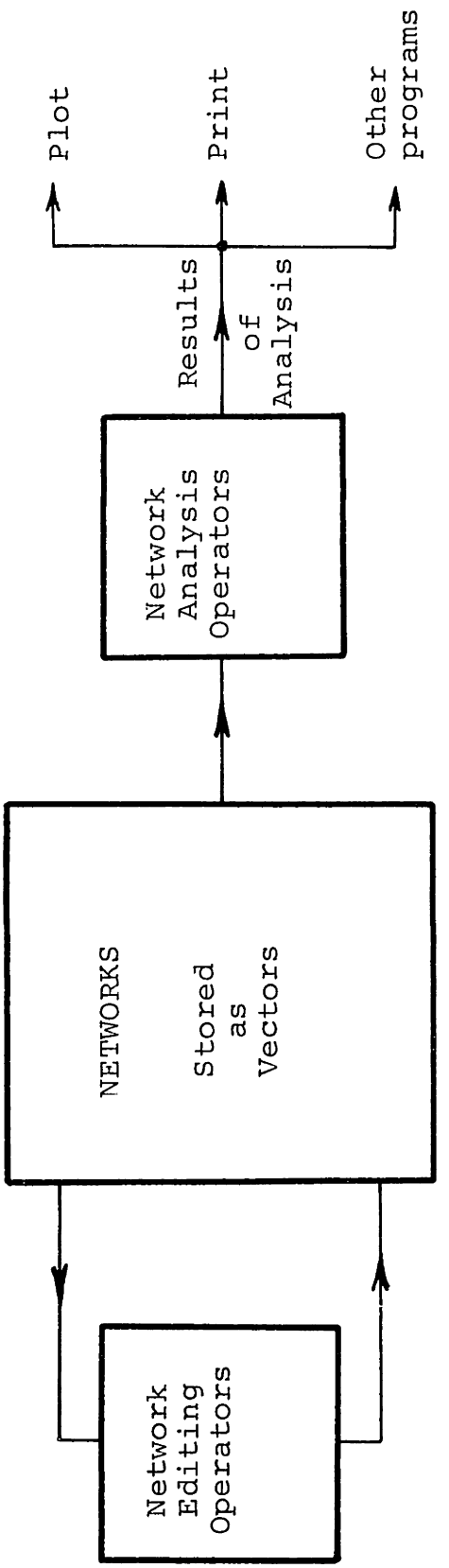


Fig. 5.1 Basic functional chart for CADNAP.

in some way, and return a new network. This new network can replace the old one or can be redefined as a new network, in which case the old unmodified network is kept unchanged. The analysis operators take a network, perform some sort of analysis on it, and return the results of that analysis. The network remains unchanged and can be used again for other analyses or modifications.

The language that CADNAP is written in is the APL language [54], [55]. This lnaguage was chosen because it permits a high degree of user/computer interaction on a time-shared basis, which is necessary for a general analysis package of this type. In particular, the array features and the operator notation of APL were found to be very desirable in the development and the use of CADNAP. A second reason for choosing the APL language was because other programs for the synthesis of digital filters are available in APL [56] and the CADNAP program serves to compliment these programs by providing a more complete overall synthesis and analysis tool for digital signal processing problems.

5.6.2 Editing Features of CADNAP

The following network editing features are presently available in CADNAP:

Branch Operators - CADNAP contains a number of branch operators which allow us to insert, delete or change any branch or set of branches in a network. They also allow us to create

new networks by inserting branches into "empty networks". The branch operators serve as a primary vehicle for building and editing networks in CADNAP.

Input/Output Operators - The input/output operators allow us to specify which nodes in a network we want to consider as input nodes and which nodes we want to consider as output nodes. They also enable us to edit networks by deleting or changing the inputs and outputs of a network. Multiple-input/multiple-output specifications for a network are allowed in CADNAP and multiple-input/multiple-output analyses can be performed.

Skeleton Operators - The skeleton operators allow us to define network topologies without specifying the coefficient values. Coefficients can be "read into" the network at a later time. This feature enables us to analyze a given network structure with many different sets of coefficients without having to completely respecify the network prior to each analysis.

Transpose Operator - The transpose operator allows us to generate a network by taking the transpose (see Chapter 3) of a given network.

Label Operators - The label operators or "asterisk" operators allow us to attach labels (asterisks) to specific branches in a network. In this way, we can identify those branches of interest for specific analysis purposes. For example, in the computation of the coefficient sensitivities of a system function, we want to specify that the sensitivities be taken

with respect to particular coefficients in the network identified by these labels.

Quantize Operators - The quantize operators allow us to quantize the coefficients within a network. This is useful when studying the practical effects of finite-precision coefficients in the implementation of the filter.

5.6.3 Analysis Features of CADNAP

The following analysis features are presently available in CADNAP:

Network Print-Out Operator - The network print-out operator is an analysis operator which takes a specified network and prints out a complete description of the network. From this description, we can completely reconstruct the network in flow graph form. This operator is particularly useful as a vehicle for user/computer interaction in the network editing phase.

Coefficient Operators - The coefficient operators permit us to "read out" specific coefficient values within a network.

The Impulse Operator - The impulse operator enables us to generate the impulse response (unit sample response) of a network for a specified duration. Computation is performed by the state space method suggested by relations (5.5) and (5.6).

The Frequency Response Operator - The frequency response is calculated on a point-by-point basis by solving two sets of

N linear simultaneous equations. The computation is performed by the method suggested in relations (5.13) and (5.14). The frequency response can be simultaneously evaluated in terms of the log magnitude, magnitude, phase, real part, and/or imaginary part of the response. It can be determined for any preselected set of frequencies.

A second frequency analysis operator enables us to compute the frequency response for the special case of cascade and ladder structures. It is especially useful for analyzing networks which are too large to effectively be handled by ordinary matrix methods. The analysis is performed by first analyzing two-input two-output subnetworks within the structure and then computing the input to output frequency response by using the chain matrix properties of ladder networks (see Chapter 2, Section 2.8).

The Sensitivity Analysis Operators - CADNAP is capable of computing coefficient sensitivities of digital networks with respect to any specific coefficients (specified by asterisks) in the network. The analysis is carried out by using the transpose network approach suggested by (5.44), (5.45), (5.46), and (5.47). Various forms of the sensitivities can be computed at specified frequencies. A second operator enables us to compute the sensitivity norm for use in the statistical wordlength definition (see Chapter 6).

Plotting Routines - Various types of plotting functions are available for plotting response functions and results in

CADNAP [38].

Other Programs in APL - Programs are available for the synthesis of Butterworth, Chebyschev, and elliptic digital filters [38], [56]. Also, numerous programs are available in APL for various numerical analysis manipulations. Because of the independence of the CADNAP operators, individual operations can be removed from the package and used separately with other programs to perform specialized analyses.

Further details on the use of the CADNAP package are available in the user's guide [38] and Appendix A.

Chapter VI

A STATISTICAL APPROACH FOR MINIMIZING THE COEFFICIENT WORD LENGTH OF A DIGITAL FILTER

6.1 INTRODUCTION

An important factor in the cost of a digital filter is the coefficient word length necessary to implement the filter. Consequently, it is desired to keep this word length to a minimum. The problem is closely associated with the choice of the filter structure and its sensitivity properties. The use of statistical methods has played an important role in the study of these coefficient sensitivity properties and the effects of coefficient quantization.

One of the first attempts to use a statistical measure for comparison of digital filters was made by Knowles and Olcayto [50]. More recently, Avenhaus [57] proposed a statistical approach for estimating the coefficient word length necessary to keep the power transfer function of a digital filter within a prescribed error bound. He demonstrated that with this approach the necessary coefficient word length could be predicted to within several bits. A similar approach has been taken by Chan and Rabiner [68] for finite impulse response filters.

In this chapter a new statistical word length is defined [49] such that the magnitude of the system function is maintained within a prescribed error bound. It is shown that the

statistical word length can be determined approximately from the evaluation of the coefficient sensitivities at only a few critical frequencies. An optimization procedure is then presented for minimizing this statistical word length for a given filter realization [49]. The statistical word length is based on the magnitude of the system function instead of the power transfer function because this choice leads to a linear expression for the statistical quantization step size which greatly simplifies the development of the optimization procedure.

6.2 A STATISTICAL ESTIMATE OF THE REQUIRED WORD LENGTH

It will be assumed in this chapter that the primary filter characteristic of interest is the magnitude of the system function. This function is designed to approximate an ideal magnitude response $H_I = H_I(\omega)$ with an allowed error deviation $\delta = \delta(\omega)$. When the filter coefficients are specified to infinite precision, the magnitude of the system function, $|T| = |T(e^{j\omega})|$, will be designated as $|T_o| = |T_o(e^{j\omega})|$. It may represent a Butterworth, Tchebyshev, or elliptic approximation or may be obtained by some algorithmic procedure. Now, let $\Delta|T| = \Delta|T(e^{j\omega})|$ be the error incurred in $|T_o|$ when finite precision coefficients are used in the actual filter. It is desired that this magnitude response, $|T| = |T_o| + \Delta|T|$, also meet the error criterion, $H_I \pm \delta$, as illustrated in Fig. 6.1.

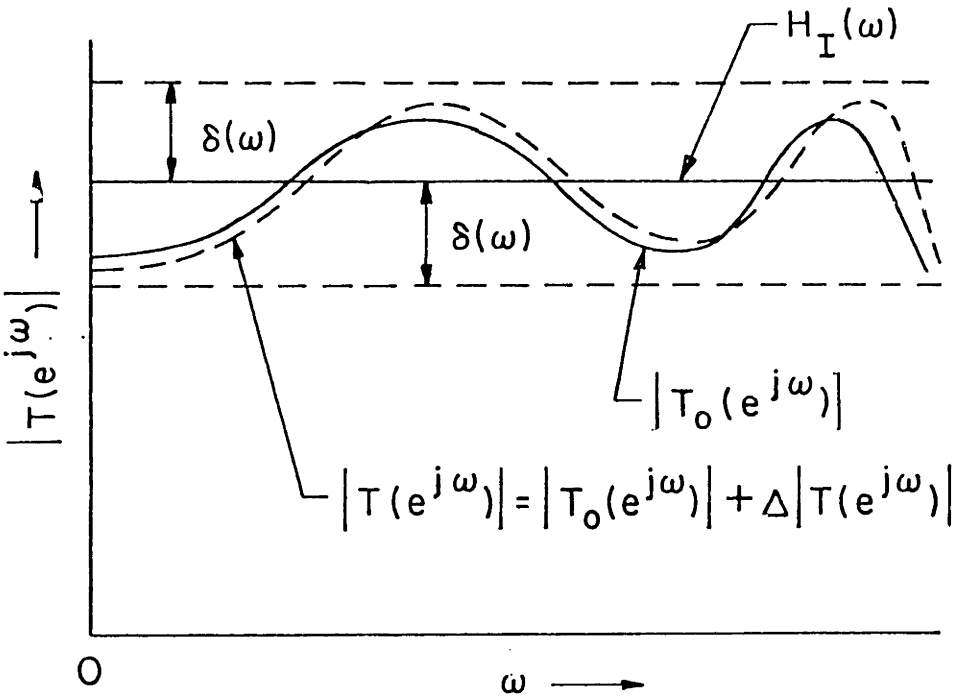


Fig. 6.1 Error criterion on the magnitude of the system function.

For a given filter structure the error, $\Delta|T|$, can be expressed in terms of a linear, first-order approximation as

$$\Delta|T| \cong \sum_{i=1}^{m} \frac{\partial|T_o|}{\partial c_i} \Delta c_i . \tag{6.1}$$

In this expression the values c_i (i=1,2,...m) represent the ideal, infinite precision coefficient values for the filter which yield the magnitude response $|T_o|$, m is the total number of coefficients, and Δc_i are the deviations of the coefficients from their ideal values due to quantization. The partial derivatives in (6.1) can be computed efficiently for arbitrary filter structures by using the transpose network approach (see chapters 3 and 5).

A statistical approach can now be taken to obtain an upper "bound" on the magnitude of the error $\Delta|T|$. It will be assumed that the quantization step size for rounding of the coefficients in fixed-point arithmetic is Q, and that the quantization errors Δc_i will have a uniform probability of lying between -Q/2 and Q/2 for an arbitrary set of coefficients. The statistical errors in the coefficients can then be characterized as having zero means, and variances of

$$\sigma^2_{\Delta c_i} = \frac{Q^2}{12} . \tag{6.2}$$

The mean of $\Delta|T|$ is then zero and the variance of $\Delta|T|$, assuming that the errors Δc_i are independent, can be determined as

$$\sigma^2_{\Delta|T|} \cong \frac{Q^2 S^2}{12} , \tag{6.3}$$

where S^2 is defined as

$$S^2 = S^2(\omega) \triangleq \sum_{i=1}^{m} \left(\frac{\partial |T_o|}{\partial c_i} \right)^2 . \tag{6.4}$$

For the case where some coefficients are quantized to greater precision than others, Q can be chosen to be the smallest quantization step size, and the corresponding quantization step sizes for the coefficients c_i can be given in terms of weighted units of this step size in the form $k_i Q$. In this case (6.4) becomes

$$S^2 = \sum_{i=1}^{m} \left(k_i \frac{\partial |T_o|}{\partial c_i} \right)^2 . \tag{6.5}$$

An approximation of the probability density function of $\Delta|T|$ can now be made following an approach similar to that of Avenhaus [57]. From the central limit theorem it can be stated that the probability density function of the sum of a number of independent random variables approaches a Gaussian distribution as the number of variables increases without limit under the condition that no particular few of the variables dominate the sum. In practice, for uniformly distributed random variables, which are approximately equally

distributed, it can be shown that, even with relatively few variables (3 or 4), the density function of the sum becomes very close to a Gaussian distribution near the mean of the sum. It will be assumed that these conditions are satisfied in defining $\Delta|T|$ to be the sum of the errors due to coefficient quantization and that the probability density function of $\Delta|T|$ can be approximated as a Gaussian distribution with zero mean and a variance of $\sigma^2_{\Delta|T|}$.

By using the Gaussian distribution assumption, it is possible to choose a value, $x\sigma_{\Delta|T|}$, such that $|\Delta|T||$ will be less than $x\sigma_{\Delta|T|}$ with probability y for an arbitrary set of coefficients. That is,

$$P[|\Delta|T|| \leq x\sigma_{\Delta|T|}] = y, \tag{6.6a}$$

or using (6.3),

$$P\left[|\Delta|T|| \leq \frac{xQS}{\sqrt{12}}\right] = y, \tag{6.6b}$$

where x and y are related by the error function,

$$y = \frac{2}{\sqrt{2\pi}} \int_0^x e^{-x^2/2}\, dx. \tag{6.7}$$

This function is plotted in Fig. 6.2 and is tabulated in numerous mathematical handbooks. Once an acceptable confidence factor, y, is selected, a corresponding value of x can be determined from Fig. 6.2 or from the tables, and a

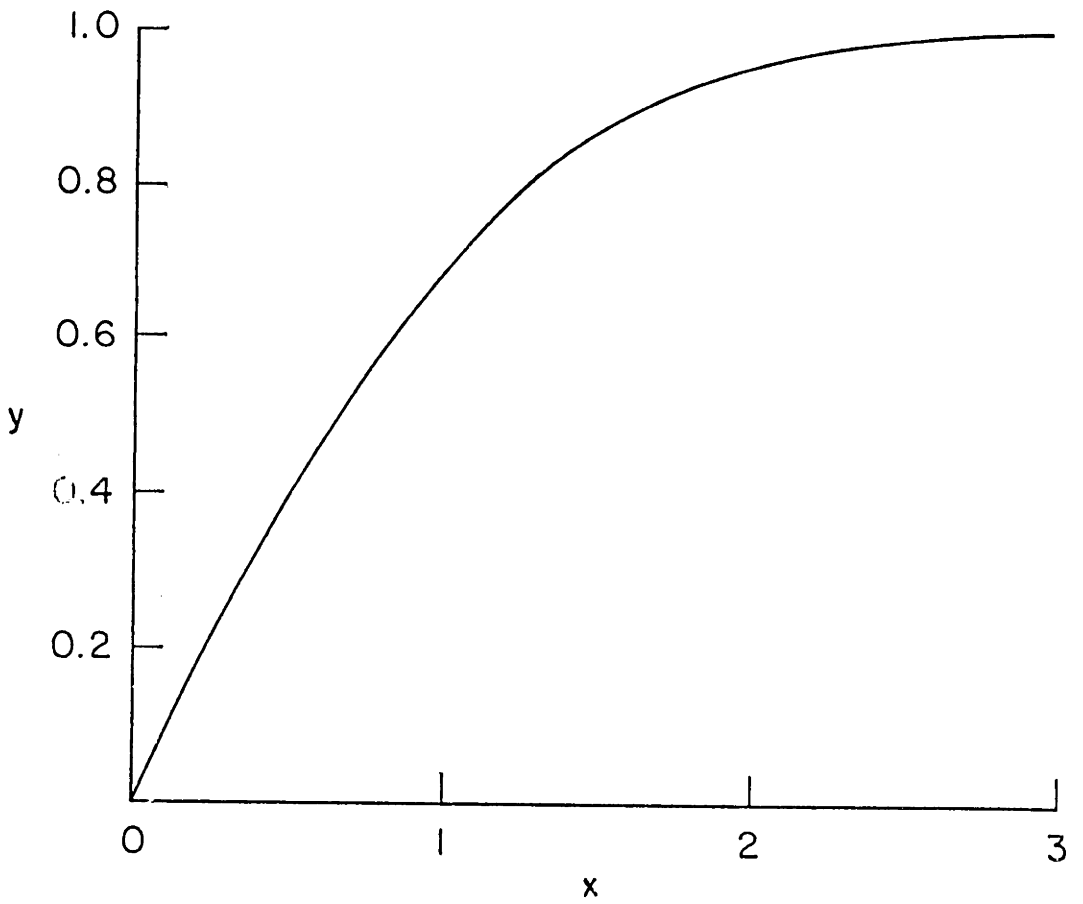


Fig. 6.2 Plot of the confidence factor y as a function of x.

statistical "bound," $x\sigma_{\Delta|T|}$, on $|\Delta|T||$ can be determined. This value, although referred to as a "bound," is not a true bound in that it is possible with probability 1-y for $|\Delta|T||$ to exceed this value.

In order to meet the specified error criterion, $H_I \pm \delta$, it is necessary to impose an upper limit on the error magnitude $|\Delta|T||$. From Fig. 6.1 it is apparent that the error criterion will be satisfied if both of the following conditions on $|\Delta|T||$ are met:

$$|\Delta|T|| \leq (H_I+\delta) - |T_o| \tag{6.8}$$

and

$$|\Delta|T|| \leq |T_o| - (H_I-\delta). \tag{6.9}$$

As the error magnitude $|\Delta|T||$ must satisfy both (6.8) and (6.9), it must be less than, or equal to, the minimum of these two conditions. This requirement can be written as a single expression by noting that the minimum of two positive numbers A and B can always be written as

$$\min (A,B) = \frac{1}{2} (|A+B|-|A-B|). \tag{6.10}$$

Using this expression, the error conditions in (6.8) and (6.9) can be expressed as

$$|\Delta|T|| \leq \delta - ||T_o|-H_I|. \tag{6.11}$$

In this chapter only piecewise constant ideal magnitude functions will be considered, such that H_I is either one or zero in prescribed regions of the frequency domain. In the passband, $H_I = 1$ and $\delta = \delta_p$. Condition (6.11) can then be expressed as

$$|\Delta|T|| \leq \delta_p - ||T_o|-1|. \tag{6.12}$$

For the stop band, $H_I = 0$ and $\delta = \delta_s$. In this case, (6.11) becomes

$$|\Delta|T|| \leq \delta_s - |T_o|. \tag{6.13}$$

In order to be assured that the filter requirements will be met under coefficient quantization with probability y, it is necessary to stipulate that the statistical "bound" on $|\Delta|T||$ satisfy the error criterion in (6.11). This gives the relationship

$$|\Delta|T|| \tilde{\leq} \frac{xQS}{\sqrt{12}} \leq \delta - ||T_o|-H_I|. \tag{6.14}$$

This condition must be satisfied for all frequencies of interest.

Now let $q(\omega)$ be the maximum allowable quantization step size that still satisfies condition (6.14) at frequency ω. This occurs when equality holds in (6.14) and $q(\omega)$ can then be expressed as

$$q(\omega) = \frac{\sqrt{12}\ (\delta - ||T_o| - H_I|)}{xS} . \qquad (6.15)$$

In order to satisfy the error criterion for all frequencies, it is necessary to specify that the actual maximum allowable quantization step size, Q, be chosen such that condition (6.14) is satisfied at all frequencies of interest. This can be accomplished by choosing Q such that

$$Q = \min_{\omega} q(\omega) = \min_{\omega} \frac{\sqrt{12}\ (\delta - ||T_o| - H_I|)}{xS} . \qquad (6.16)$$

The coefficient word length of the digital filter can now be defined as

$$W = 1 + i_M - i_L, \qquad (6.17)$$

where 2^{i_M} is the power of 2 represented by the most significant bit and 2^{i_L} is the power of 2 represented by the least significant bit. This definition does not include the extra bit needed for representing the sign. The most significant bit, i_M, must be chosen such that all of the coefficient values lie in the range

$$-2 \cdot 2^{i_M} < c_i < 2 \cdot 2^{i_M}. \qquad (6.18)$$

The least significant bit i_L is chosen to be the log to the base 2 of the largest allowable quantization step size. For

the definition of the statistical word length, it is chosen to be

$$i_L = \log_2 Q. \tag{6.19}$$

The statistical word length as a function of frequency can also be defined with the aid of (6.15) as

$$w(\omega) = 1 + i_M - \log_2 \left[\frac{\sqrt{12}\ (\delta - ||T_o| - H_I|)}{xS} \right]. \tag{6.20}$$

To be assured of meeting the error criterion of the filter for all frequencies of interest with confidence factor y, it is necessary to define the statistical word length W as

$$W = \max_{\omega} w(\omega). \tag{6.21}$$

Alternatively, it can be derived with the aid of (6.16) and (6.19) as

$$W = 1 + i_M - \log_2 \left[\min_{\omega} \frac{\sqrt{12}\ (\delta - ||T_o| - H_I|)}{xS} \right] \tag{6.22a}$$

or

$$W = 1 + i_M + \log_2 \left[\max_{\omega} \frac{xS}{\sqrt{12}\ (\delta - ||T_o| - H_I|)} \right]. \tag{6.22b}$$

This statistical word length is somewhat different than that proposed by Avenhaus [57], although both definitions

yield approximately the same result. Rewritten in terms of the notation in this paper, and including the generalization to arbitrary confidence factors, the statistical word length as a function of ω, as proposed by Avenhaus, is given as

$$w(\omega) = 1 + i_M - \log_2 \left[\frac{\sqrt{12}\ |-2H_I\delta(1-\max_{\omega}\epsilon(\omega))+\delta^2(1-\max_{\omega}^2\epsilon(\omega))|}{x\hat{S}} \right],$$

where

$$\epsilon(\omega) = \begin{cases} \frac{1}{\delta_p}\ |1-|T_o|| & \text{PB} \\ \frac{1}{\delta_s}\ |T_o| & \text{SB} \end{cases}$$

and

$$\hat{S} = \left(\sum_{i=1}^{m} \left(\frac{\partial |T_o|^2}{\partial c_i} \right)^2 \right)^{1/2} .$$

The difference between this expression and (6.20) is demonstrated by the bandpass filter example in Fig. 6.3. The magnitude $|T_o|$ in the passband ripples between $1 + \varepsilon_p$ and $1 - \varepsilon_p$ where $\varepsilon_p = 0.0157816$. The allowed passband error deviation, δ_p, was selected to be $2\varepsilon_p$ and y was chosen to be 0.95. In Fig. 6.3(b), the statistical word length $w(\omega)$ is plotted using both definitions for the case of a cascade realization. It can be observed from Fig. 6.3 that the statistical word length $w(\omega)$ defined in (6.20) has its maxima near the frequencies for which $|T_o|$ has its maximum error, $1 \pm \varepsilon_p$. This

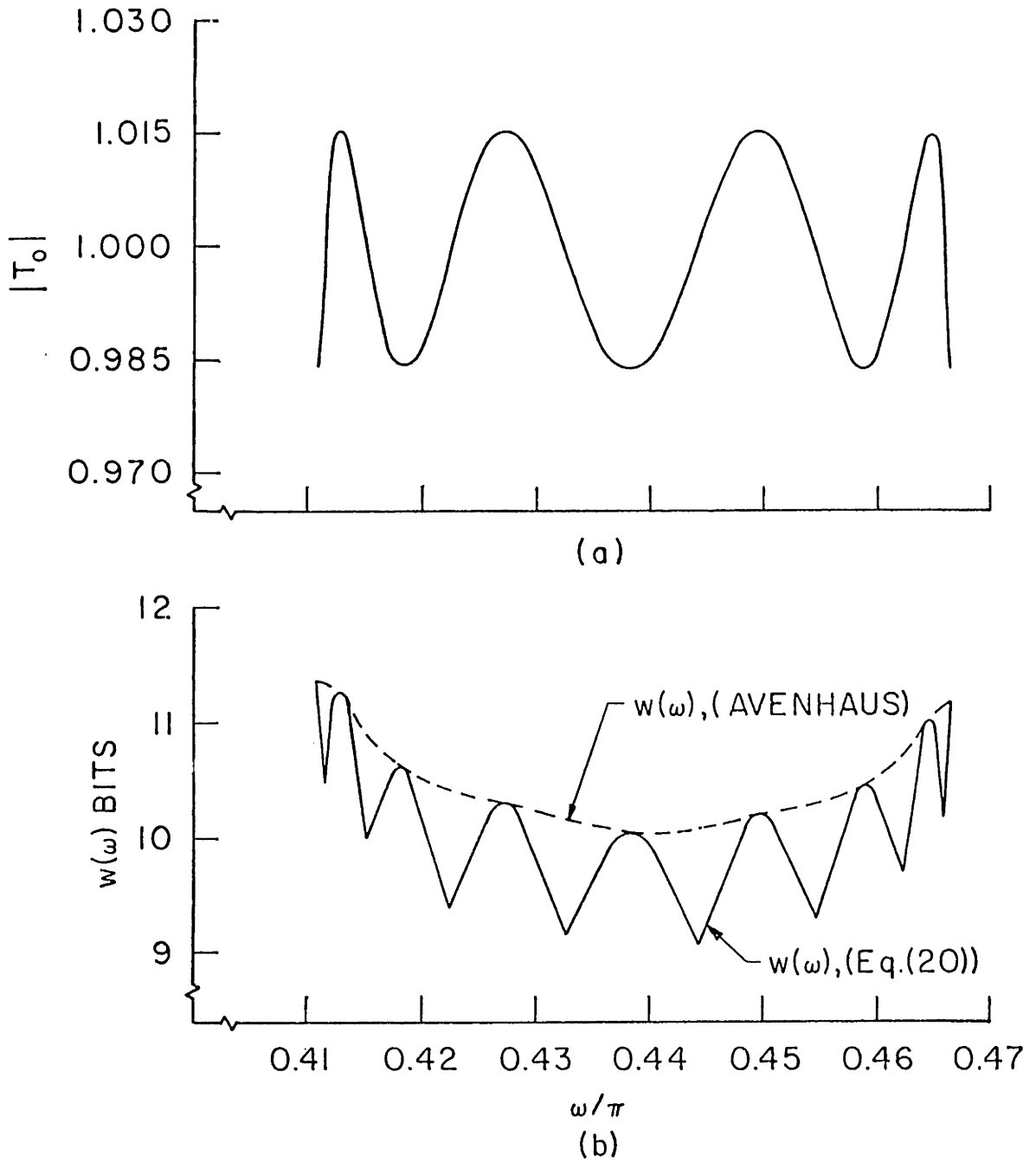


Fig. 6.3 (a) Passband frequency response for the bandpass filter example.

(b) Statistical word length for a cascade filter realization with $\delta_p = 2\varepsilon_p$ and y = 0.95, (x=2).

property is extremely useful in computing W in (6.21) as it implies that $w(\omega)$ has to be computed at only a relatively small set of frequencies in order to determine the maximum, W, rather than at all of the frequencies. This property will be exploited more fully in the next section and several examples will be given. The same behavior can be observed in the definition by Avenhaus if $\max_{\omega} \epsilon(\omega)$ is replaced by $\epsilon(\omega)$.

6.3 A PERTURBATIONAL APPROXIMATION IN THE STATISTICAL APPROACH

We have noted in the example in Fig. 6.3 and other examples of equiripple filters, that $w(\omega)$, defined in (6.20), achieves its maxima near the maximum error frequencies of $|T_o|$. This is because $S(\omega)$ appears, on the basis of experimental observations, to be a relatively slow-varying function of ω compared with the ripple behavior $\delta - ||T_o| - H_I|$ (see Fig. 6.3). For the case of filters with monotonic magnitude responses this property may not hold. Consequently, the discussion in this section will be limited to the case of equiripple filters. The statistical word lengths necessary for the passband and stop band will be derived assuming the slow-varying property of $S(\omega)$. The overall statistical word length for the filter is then chosen as the maximum of these two word lengths.

Consider first the magnitude response in the stop band (see Fig. 6.4). At the set of stop band frequencies

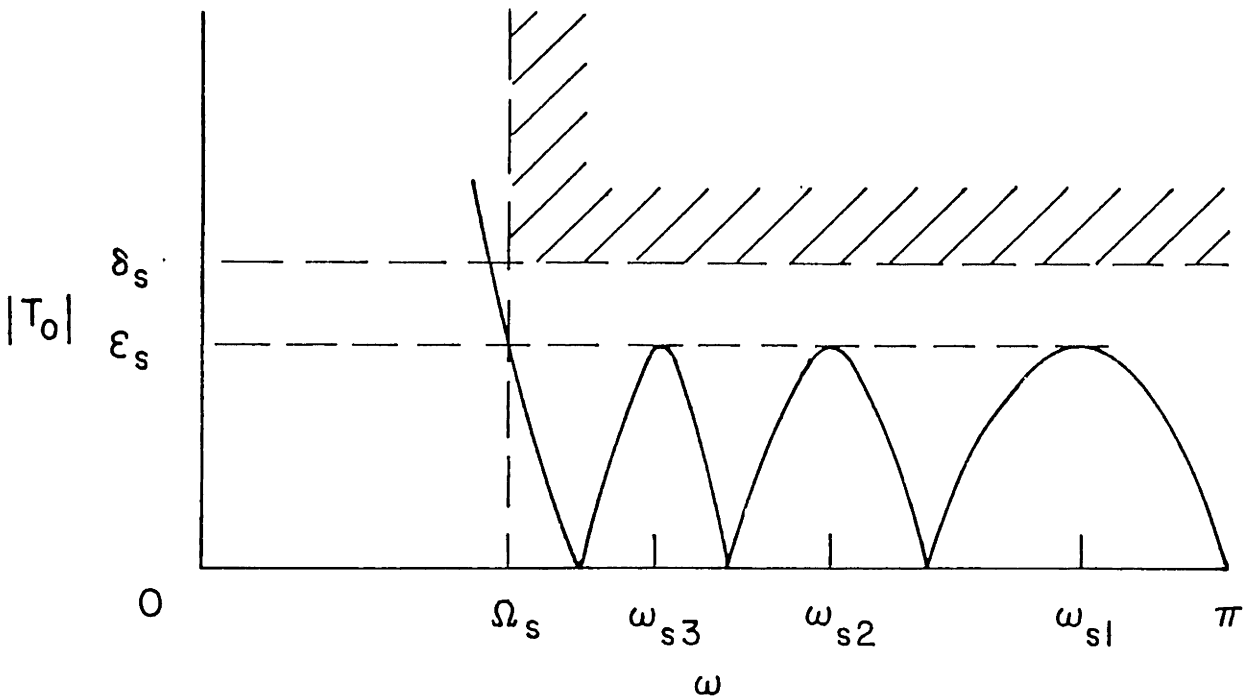


Fig. 6.4 Stop band magnitude response.

$\omega_s = \{\Omega_s, \omega_{s1}, \omega_{s2}, \ldots\}$, the magnitude $|T_o|$ equals its maximum value ε_s. The allowed error deviation in the stop band, due to coefficient quantization, is δ_s where $\delta_s > \varepsilon_s$. If the stop band constraints are tightened such that $\delta_s - \varepsilon_s$ approaches zero, then it is apparent that $W(\omega)$ will have its maxima at the maximum magnitude frequencies ω_s. Under this perturbational assumption, the maximum quantization step size, Q_s, necessary to meet the stop band requirements, can be derived with the aid of (6.13), (6.14), and (6.16) as

$$Q_s = \min_{\omega_s} \frac{\sqrt{12}\ (\delta_s - |T_o|)}{xS(\omega_s)} \tag{6.23a}$$

or

$$Q_s = \frac{\sqrt{12}\ (\delta_s - \varepsilon_s)}{x \max\limits_{\omega_s} S(\omega_s)} . \tag{6.23b}$$

The minimum required statistical word length, W_s, necessary to meet the stop band constraints can then be given as

$$W_s = 1 + i_M + \log_2 \left[\frac{x \max\limits_{\omega_s} S(\omega_s)}{\sqrt{12}\ (\delta_s - \varepsilon_s)} \right] . \tag{6.24}$$

This perturbational approximation appears, on the basis of several examples, to hold sufficiently well in practice for estimating W_s, even when the constraint, δ_s, is relaxed to the point where $\delta_s \approx 2\varepsilon_s$. Some examples will be given later.

In the passband a similar approximation can be made. In this case, $|T_o|$ alternates between $1 + \varepsilon_p$ and $1 - \varepsilon_p$, and it must satisfy both an upper and a lower bound. In addition, if the coefficient sensitivity is much larger or smaller at one set of bound frequencies than at the other, a better estimate of the statistical word length can be made if scaling is allowed. For the sake of simplicity, the scaling will be made on the ideal magnitude response H_I rather than on the system function $|T_o|$. This passband situation is illustrated in Fig. 6.5 where the ideal magnitude, H_I, is scaled to the value KH_I.

From the definitions in Fig. 6.5, we can state the following relations,

$$\delta_p + \Delta_k = \varepsilon_p + \Delta_u, \tag{6.25}$$

$$\delta_p - \Delta_k = \varepsilon_p + \Delta_\ell, \tag{6.26}$$

and

$$KH_I = H_I + \Delta_k. \tag{6.27}$$

A perturbational assumption can now be made in the passband. If the passband constraints are tightened such that $\delta_p - \varepsilon_p$ approaches zero, the smallest error constraints on $|\Delta|T||$ will occur at the frequencies for which $|T_o|$ has its maximum error of $1 \pm \varepsilon_p$. These frequencies are depicted in

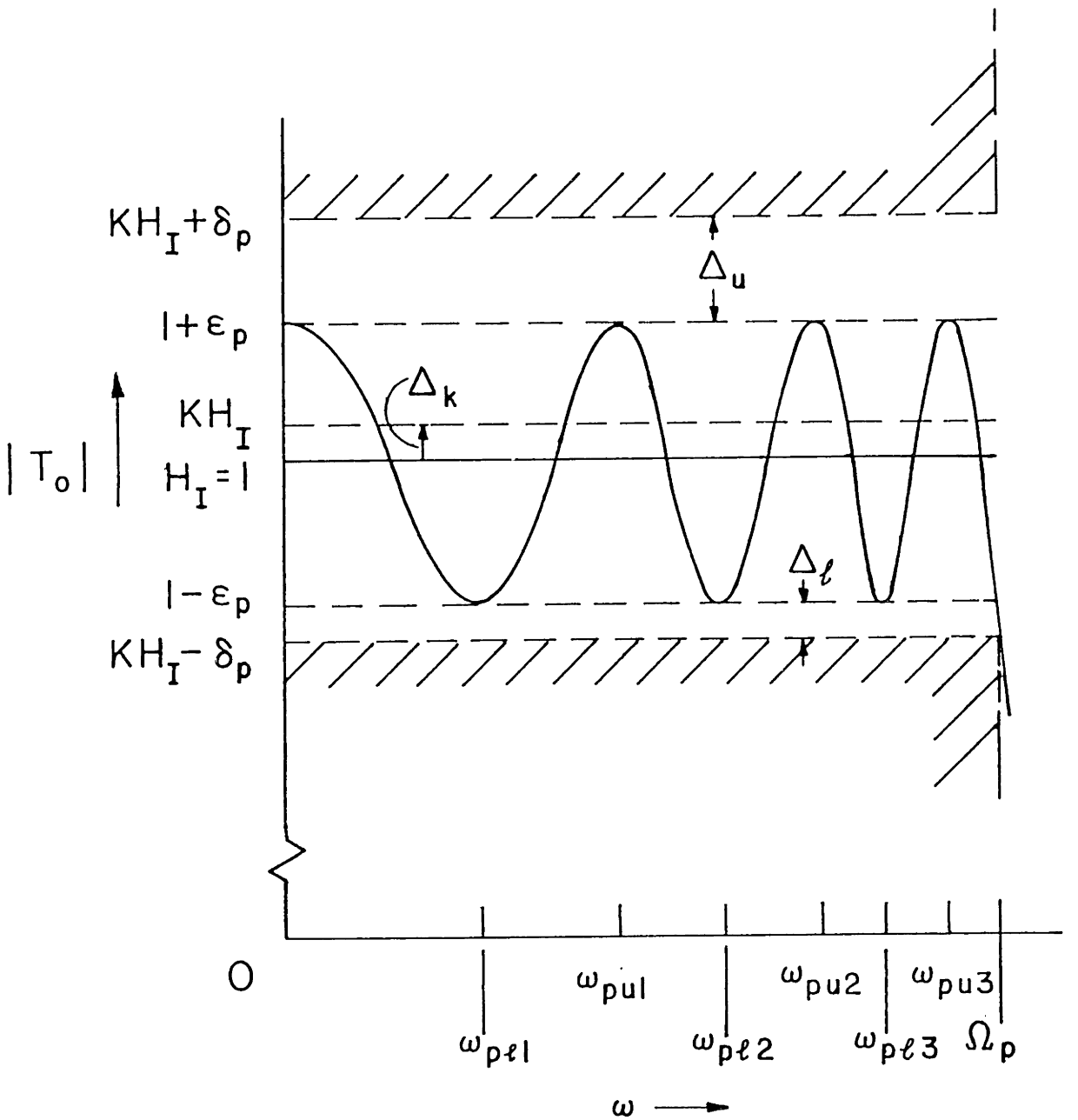


Fig. 6.5 Passband magnitude response.

Fig. 6.5 as $\omega_{pu} = \{0, \omega_{pu1}, \omega_{pu2}, \ldots\}$ for the upper bound and $\omega_{p\ell} = \{\omega_{p\ell 1}, \omega_{p\ell 2}, \ldots, \Omega_p\}$ for the lower bound. At the upper bound frequencies, ω_{pu}, the corresponding error constraint on $||\Delta|T||$ is

$$||\Delta|T|| \leq \Delta_u. \tag{6.28}$$

Similarly, the error constraint at the lower bound frequencies, $\omega_{p\ell}$, is

$$||\Delta|T|| \leq \Delta_\ell. \tag{6.29}$$

By the same argument used to develop (6.16), the corresponding quantization step sizes Q_{pu} and $Q_{p\ell}$ necessary to satisfy the upper bound constraint (6.28) and the lower bound constraint (6.29), respectively, can be expressed as

$$Q_{pu} = \min_{\omega_{pu}} \frac{\sqrt{12}\,\Delta_u}{xS(\omega_{pu})} = \frac{\sqrt{12}\,\Delta_u}{x \max_{\omega_{pu}} S(\omega_{pu})}, \tag{6.30}$$

and

$$Q_{p\ell} = \min_{\omega_{p\ell}} \frac{\sqrt{12}\,\Delta_\ell}{xS(\omega_{p\ell})} = \frac{\sqrt{12}\,\Delta_\ell}{x \max_{\omega_{p\ell}} S(\omega_{p\ell})}. \tag{6.31}$$

A tradeoff between Q_{pu} and $Q_{p\ell}$ can be made by varying the scale factor K. The quantization step size Q_{pu} decreases as

K increases, and $Q_{p\ell}$ increases as K increases. The overall quantization step size Q_p necessary to meet both constraints (6.28) and (6.29) in the passband simultaneously, must equal the minimum of Q_{pu} and $Q_{p\ell}$. As we wish to minimize the statistical word length, the scale factor K should be chosen to maximize Q_p. This occurs when K is chosen such that

$$Q_p = Q_{pu} = Q_{p\ell}. \tag{6.32}$$

With the aid of (6.25), (6.26), (6.30), (6.31), and (6.32), the quantization step size Q_p can now be derived as

$$Q_p = \frac{\sqrt{12}\ (\delta_p - \varepsilon_p)}{\frac{x}{2}\left[\max_{\omega_{pu}} S(\omega_{pu}) + \max_{\omega_{p\ell}} S(\omega_{p\ell})\right]}. \tag{6.33}$$

The corresponding statistical word length, W_p, necessary to meet the passband constraints can be given as

$$W_p = 1 + i_M + \log_2\left[\frac{\frac{x}{2}\left[\max_{\omega_{pu}} S(\omega_{pu}) + \max_{\omega_{p\ell}} S(\omega_{p\ell})\right]}{\sqrt{12}\ (\delta_p - \varepsilon_p)}\right]. \tag{6.34}$$

Finally, the scale factor K necessary to achieve the condition in (6.32) can be determined using (6.25), (6.26), (6.30), (6.31), and (6.32) and noting that $H_I = 1$ in the passband. It can be expressed in the form

$$K = 1 + (\delta_p - \varepsilon_p) \cdot \frac{\max_{\omega_{pu}} S(\omega_{pu}) - \max_{\omega_{p\ell}} S(\omega_{p\ell})}{\max_{\omega_{pu}} S(\omega_{pu}) + \max_{\omega_{p\ell}} S(\omega_{p\ell})} . \qquad (6.35)$$

It can be noted that this choice of K will always be within the range

$$1 - (\delta_p - \varepsilon_p) \leq K \leq 1 + (\delta_p - \varepsilon_p). \qquad (6.36)$$

Use of this scale factor can allow, at most, one bit of improvement in the estimate of the statistical word length. This can occur for some special classes of structures such as those synthesized from insertion-loss analog designs [15], [16], [17], [58].

Examples

To verify the assumptions made in arriving at (6.24) and (6.34) and to demonstrate the use of the statistical word length in comparing structures on the basis of sensitivity, the bandpass filter example in Fig. 6.3(a) was chosen. The parameters of the filter are:

$\varepsilon_p = 0.0157816$, $\varepsilon_s = 0.00526063$,

$\Omega_{p1} = 0.4111111\ \pi$, $\Omega_{s1} = 0.3847015\ \pi$,

$\Omega_{p2} = 0.4666667\ \pi$, $\Omega_{s2} = 0.4944444\ \pi$.

Five different structures were realized to implement this system function. The statistical word length was

computed in terms of the relative error, R, where R is defined in the passband and the stop band, respectively, as

$$R = R_p = \frac{\delta_p - \varepsilon_p}{\varepsilon_p} \quad \text{(passband)} \tag{6.37}$$

and

$$R = R_s = \frac{\delta_s - \varepsilon_s}{\varepsilon_s} \quad \text{(stop band)}. \tag{6.38}$$

The coefficients in the different implementations were then quantized by rounding in fixed-point arithmetic and the corresponding errors in the magnitude of the system function were computed. These errors were expressed in terms of the relative error criterion with δ_p and δ_s in this case representing the actual maximum passband and stop band errors, respectively. Scaling was used such that the actual passband error extremes corresponded to $1 \pm \delta_p$. These results were plotted against the statistical results and are presented in Fig. 6.6.

The first three examples correspond to the cascade structure, the direct form structure, and the parallel form structure, respectively. The fourth example, Fig. 6.6(d), corresponds to a continued fraction expansion realization [31] of type 1B and the fifth example, Fig. 6.6(e), corresponds to a continued fraction expansion (ladder) realization [18] in which the poles are realized by the continued fraction structure and the zeros are obtained from a linear combination of the state variables.

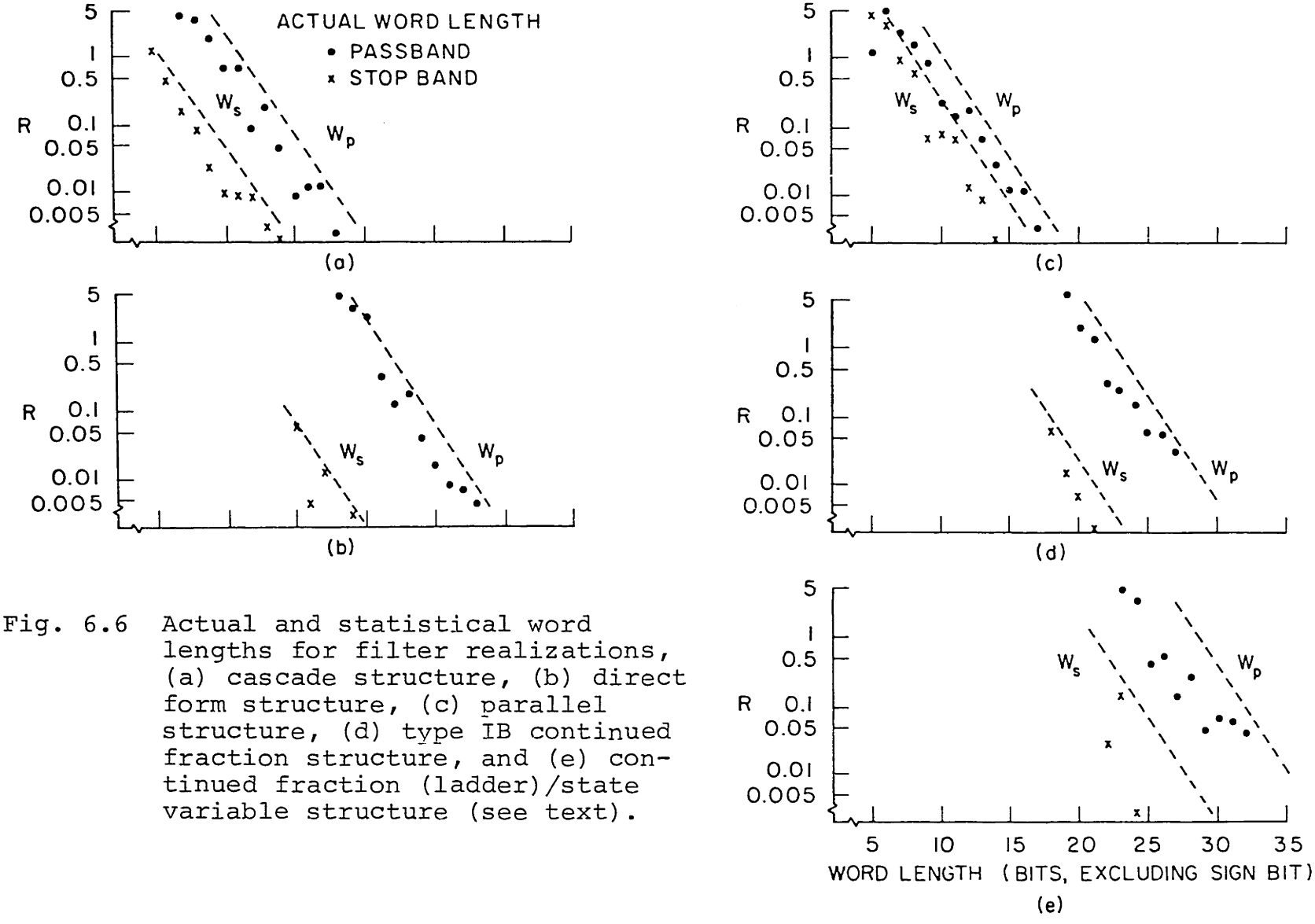


Fig. 6.6 Actual and statistical word lengths for filter realizations, (a) cascade structure, (b) direct form structure, (c) parallel structure, (d) type IB continued fraction structure, and (e) continued fraction (ladder)/state variable structure (see text).

In all of the results the statistical word length appears to be a reasonable "bound" on the actual required word length. To compute the statistical word lengths for the plots in Fig. 6.6, the coefficient sensitivity of each network had to be computed at nine frequencies in the passband and six frequencies in the stop band, with two network analyses required at each frequency. In comparison, computing each point of the actual error under coefficient quantization required analysis of each network at approximately 125 frequencies in the passband and 125 frequencies in the stop band. This represents a considerable amount of computation for general sensitivity comparisons of structures since many such points must be computed to generate plots such as those shown in Fig. 6.6. Consequently, for such comparisons, the statistical word length approach appears to be an efficient and useful criterion. In the filter example of Fig. 6.6, for a specified relative error, R_p, with the statistical word length criterion, the direct form structure requires approximately 10 bits greater coefficient accuracy than the cascade structure; the parallel structure requires approximately 1 bit less than the cascade structure; the continued fraction structure of Fig. 6.6(d) requires approximately 11 bits more; and the continued fraction structure of Fig. 6.6(e) requires approximately 17 bits more than the cascade structure.

6.4 AN OPTIMIZATION TECHNIQUE FOR MINIMIZING THE STATISTICAL WORD LENGTH

Generally, in a filter design problem with a piecewise constant magnitude response, such as a lowpass filter, δ_p, δ_s, Ω_p, and Ω_s are prespecified, where δ_p and δ_s are the allowed passband and stop band errors, respectively, and Ω_p and Ω_s are the passband and stop band edge frequencies, respectively. From these specifications, the necessary filter order is calculated. Since the actual order of the filter must be an integer, the next largest integer, n, is then selected as the required order of the filter. Because of this selection, there exists a continuum of approximations, $\{|T|\}$, with passband and stop band errors ε_p and ε_s, respectively, which satisfy the constraints $\delta_p \geq \varepsilon_p$ and $\delta_s \geq \varepsilon_s$. The question then arises as to which approximation in this set should be chosen such that the required coefficient word length for a given filter structure is minimized. In general, for the case of digital filters, this is a nonlinear integer programming problem and a solution is difficult to obtain.

In this chapter a different approach is taken. Instead of minimizing the actual coefficient word length, we determine the optimal infinite precision approximation, $|T_o|$, which minimizes the statistical word length for a given structure. While this choice of $|T_o|$ does not guarantee that the actual coefficient word length will be minimized upon rounding of the coefficients, it has been observed in examples that the

approach generally leads to a reduction of the actual coefficient word length. The philosophy of the optimization procedure is to choose the approximation $|T_o|$ such that it is specifically matched to the coefficient sensitivity properties of the structure. That is, we choose $|T_o|$ such that at frequencies where the coefficient sensitivity is high there is greater allowance for error and where it is low there is less allowance for error. Although we will specifically concentrate on piecewise-constant lowpass filter designs in this chapter, the concept extends to other designs as well.

6.4.1 Two-Parameter Statistical Approach

To demonstrate the statistical word length minimization procedure, consider first the two-parameter approach in which ε_p and ε_s are allowed to vary and the passband and stop band edge frequencies are fixed. If ε_p is decreased, the corresponding passband statistical word length W_p will decrease. This is done at the expense of increasing ε_s and the stop band statistical word length, W_s. The overall coefficient word length, W, of the digital filter must equal the maximum of W_p and W_s in order to satisfy both the passband and stop band constraints. As we desire to minimize W, the obvious choice is to specify ε_p and ε_s such that

$$W = W_p = W_s, \tag{6.39}$$

or equivalently,

$$Q = Q_p = Q_s. \tag{6.40}$$

With the aid of (6.23b) and (6.33), the necessary relationship between ε_p and ε_s, in order to satisfy (6.40), can be derived as

$$\frac{\delta_s - \varepsilon_s}{\delta_p - \varepsilon_p} = B, \tag{6.41}$$

where

$$B \overset{\Delta}{=} \frac{\max\limits_{\omega_s} S(\omega_s)}{\frac{1}{2}\left|\max\limits_{\omega_{pu}} S(\omega_{pu}) + \max\limits_{\omega_{p\ell}} S(\omega_{p\ell})\right|}. \tag{6.42}$$

This expression is valid for monotonic response functions as well as equiripple responses. In the monotonic case, however, extra frequency points may have to be included in the sets ω_s, ω_{pu}, and $\omega_{p\ell}$ to assure that the proper maxima of S are obtained. Another point to be noted in the above relationship is that the factor x cancels, making the optimization procedure independent of the choice of the confidence factor. Consequently, for any choice of y the same optimal infinite precision system function will be obtained for realizing the minimum statistical word length.

A second relationship between ε_p and ε_s can be obtained from the approximation used in determining $|T_o|$. This relationship is dependent on the type of approximation chosen.

By constraining the passband and stop band edge frequencies to be fixed, an intrinsic nonlinear relationship between ε_p and ε_s can be expressed in the form

$$F(\varepsilon_p, \varepsilon_s) = 0. \tag{6.43}$$

For the case of analytic approximations such as the Butterworth, Tchebyshev, or elliptic approximations, analytic expressions for (6.43) are generally available. For algorithmic designs the intrinsic relation in (6.43) is implied but an analytic expression may not be available. In this case the intrinsic function may have to be determined numerically over the region of interest.

The objective of the optimization procedure is to solve equations (6.41) and (6.43) for the two unknowns ε_p and ε_s. The procedure begins by computing an initial approximation $|T_o|_1$ which satisfies the passband and stop band constraints δ_p and δ_s. This sets the order of the filter, n, and gives an initial trial solution $\varepsilon_{p,1}$ and $\varepsilon_{s,1}$. An appropriate filter structure is chosen for the realization, and the sensitivity ratio B_1 for this first trial is computed using (6.42). The solution to (6.41) and (6.43) is then obtained by an iterative procedure. On each trial, j, of the procedure the sensitivity ratio B_j is assumed to be invariant over the continuum of feasible solutions $(\varepsilon_p, \varepsilon_s)$, where the feasible solutions are defined to be the solutions which satisfy the constraints δ_p and δ_s and the intrinsic relation (6.43). Using this

assumption, we choose the value of B in (6.41) to be B_j and the new trial values $(\varepsilon_{p,j+1}, \varepsilon_{s,j+1})$ are determined by solving (6.41) and (6.43) for $(\varepsilon_p, \varepsilon_s)$. A new filter is synthesized with the parameters $(\varepsilon_{p,j+1}, \varepsilon_{s,j+1})$, using the same structure as that of the initial trial, and a new sensitivity ratio B_{j+1} is computed from (6.42) for this realization. This new sensitivity ratio, B_{j+1}, should be an improved estimate of the actual value of B. The procedure is then repeated for trials j+1, j+2, ... until it converges to the optimal solution which satisfies (6.41), (6.42), and (6.43) simultaneously, thus minimizing the statistical word length. The convergence of the procedure was found, on the basis of empirical observations, to be very rapid. For the examples analyzed, satisfactory solutions were generally obtained after three or four trials. A block diagram illustrating the above optimization procedure is given in Fig. 6.7.

For the two-parameter approach, it can be shown, with minor assumptions on $F(\varepsilon_p, \varepsilon_s)$, that the solutions $(\varepsilon_{p,j+1}, \varepsilon_{s,j+1})$ obtained in the trials are guaranteed to be feasible solutions. It will be assumed that the intrinsic relation $F(\varepsilon_p, \varepsilon_s)$ is of the form such that in the $(\varepsilon_p, \varepsilon_s)$ parameter space, ε_s can be expressed as a single-valued, monotonically decreasing function of ε_p such that as ε_p approaches zero, ε_s approaches infinity, and as ε_s approaches zero, ε_p approaches infinity. Because of the choice of the initial trial, there exists at least one point $(\varepsilon_{p,1}, \varepsilon_{s,1})$ on this curve that is

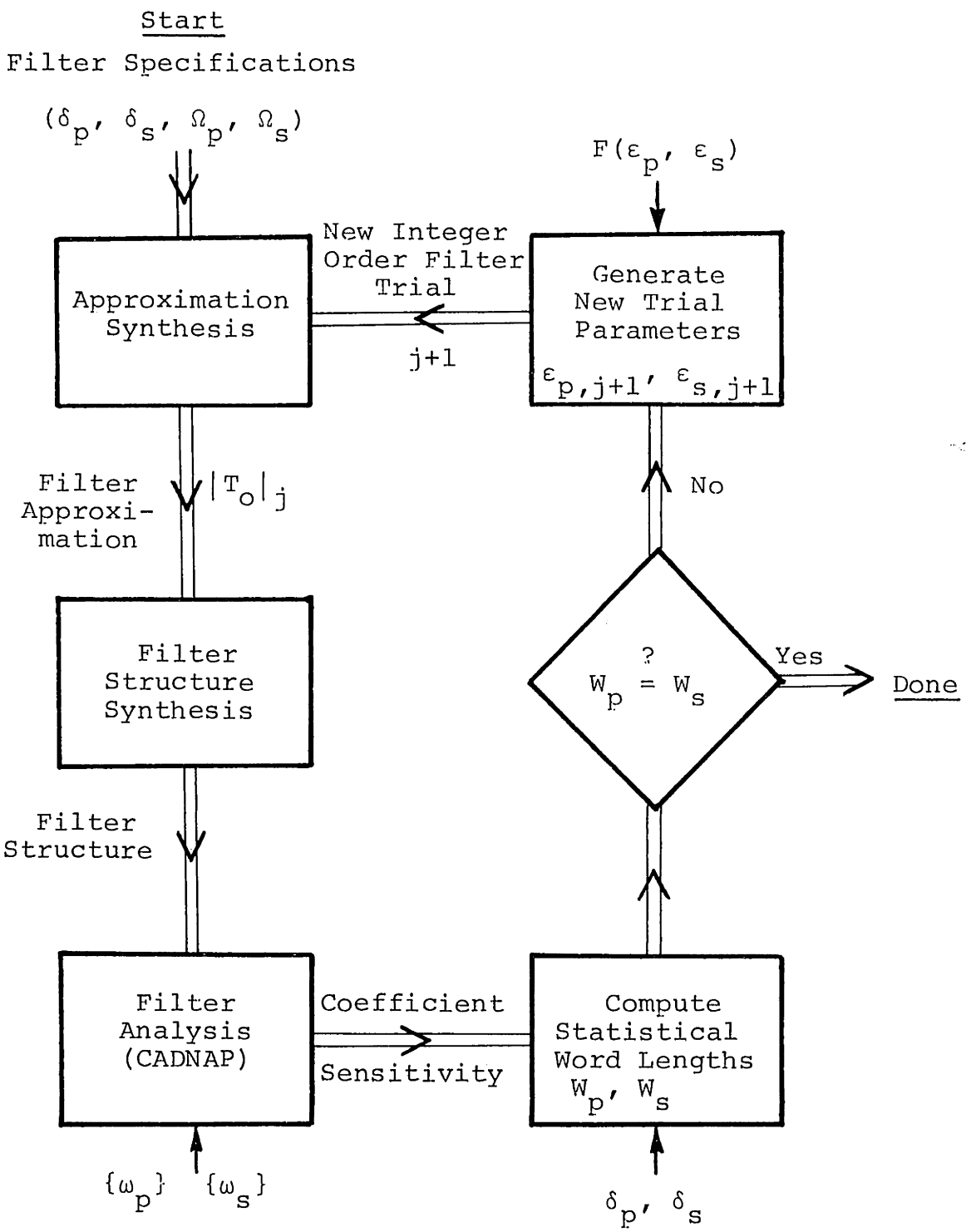


Fig. 6.7 Block diagram for the statistical word length optimization procedure.

a feasible solution, that is, $\varepsilon_{p,1} \leq \delta_p$ and $\varepsilon_{s,1} \leq \delta_s$. It can also be observed that all of the values of B_j obtained from (6.42) are necessarily positive. Consequently the ratio $(\delta_s-\varepsilon_{s,j+1})/(\delta_p-\varepsilon_{p,j+1})$ must be positive by relation (6.42). For any $\varepsilon_{p,j+1}$ such that $\varepsilon_{p,j+1} \geq \varepsilon_{p,1}$, the corresponding $\varepsilon_{s,j+1}$ on the curve, $F(\varepsilon_p, \varepsilon_s) = 0$, must satisfy the condition $\varepsilon_{s,j+1} \leq \varepsilon_{s,1}$ by the assumptions made on $F(\varepsilon_p, \varepsilon_s)$. Therefore, $(\delta_s-\varepsilon_{s,j+1})$ is positive and, from (6.41), the value $(\delta_p-\varepsilon_{p,j+1})$ must be positive because the ratio was shown to be positive. Consequently, $\varepsilon_{p,j+1} \leq \delta_p$ and feasibility is assured. A similar argument can be made for the case when $\varepsilon_{p,j+1} \leq \varepsilon_{p,1}$. Thus, all of the trials result in feasible solutions. A general proof for convergence of these trials was not found because of the highly complex nature of B which is dependent on the type of structure, the type of approximation, and the specifications of the filter.

Examples

To demonstrate the method, an elliptic lowpass filter was chosen. For the elliptic approximation, the intrinsic function $F(\varepsilon_p, \varepsilon_s)$ is well known [2], [59], [60] and can be written in the form

$$F(\varepsilon_p, \varepsilon_s) = \frac{K(k_{1c})\ K(k)}{K(k_1)\ K(k_c)} - n = 0, \qquad (6.44)$$

where

n = the order of the filter,

$K(\cdot)$ = the complete elliptic integral of the first kind,

k = the selectivity factor

$$= \frac{\tan\ (\Omega_{pc}/2)}{\tan\ (\Omega_{sc}/2)}\ , \tag{6.45}$$

$$k_c = +\sqrt{1 - k^2},$$

$$k_1 = \sqrt{\frac{\left(\frac{1 + \varepsilon_p}{1 - \varepsilon_p}\right)^2 - 1}{\left(\frac{1 + \varepsilon_p}{\varepsilon_s}\right)^2 - 1}}\ , \tag{6.46}$$

and

$$k_{1c} = +\sqrt{1 - k_1^2}.$$

For the two-parameter statistical approach, the elliptic design edge frequencies Ω_{pc} and Ω_{sc} are assigned to be equal to the specified passband and stop band edge frequencies. That is, $\Omega_{pc} = \Omega_p$ and $\Omega_{sc} = \Omega_s$.

In order to compute the values B, W_p, and W_s efficiently, we should know the maximum error frequencies ω_{pu}, $\omega_{p\ell}$, and ω_s. These frequencies can be determined, in the case of the elliptic approximation, from the elliptic functions; expressions

for their evaluation are given in Section 6.5.

To obtain the successive trial parameters it is necessary to solve (6.41) and (6.44) with $B = B_j$. This can be accomplished by expressing (6.41) in the parametric form

$$\varepsilon_{s,j+1} = \varepsilon_s = B_j t + \delta_s, \tag{6.47a}$$

$$\varepsilon_{p,j+1} = \varepsilon_p = t + \delta_p, \tag{6.47b}$$

where t is a dummy variable. Upon substitution in (6.44), the intrinsic expression $F(\varepsilon_p, \varepsilon_s)$ can be written in terms of the single variable t as $F(t) = 0$. Consequently, the problem is reduced to the simpler problem of determining a zero of a nonlinear one-dimensional function in t. Many elegant algorithms exist for solving this problem but as F(t) is well behaved in the region of interest, the following simple approach, sometimes referred to as the bisection method [9], works well and does not require the evaluation of derivatives.

The problem can be clearly illustrated in the ε_p, ε_s parameter space (see Fig. 6.8). The parametric equations (6.47a) and (6.47b) trace a straight line in this space with positive slope B_j. The intrinsic relation traces a curve of feasible solutions over the region $\varepsilon_p \leq \delta_p$, $\varepsilon_s \leq \delta_s$. Above this curve $F(\varepsilon_p, \varepsilon_s) < 0$ and below it $F(\varepsilon_p, \varepsilon_s) > 0$. The desired solution is the intersection of the curve and the straight line. Choosing $t = t_a = 0$ corresponds to the point

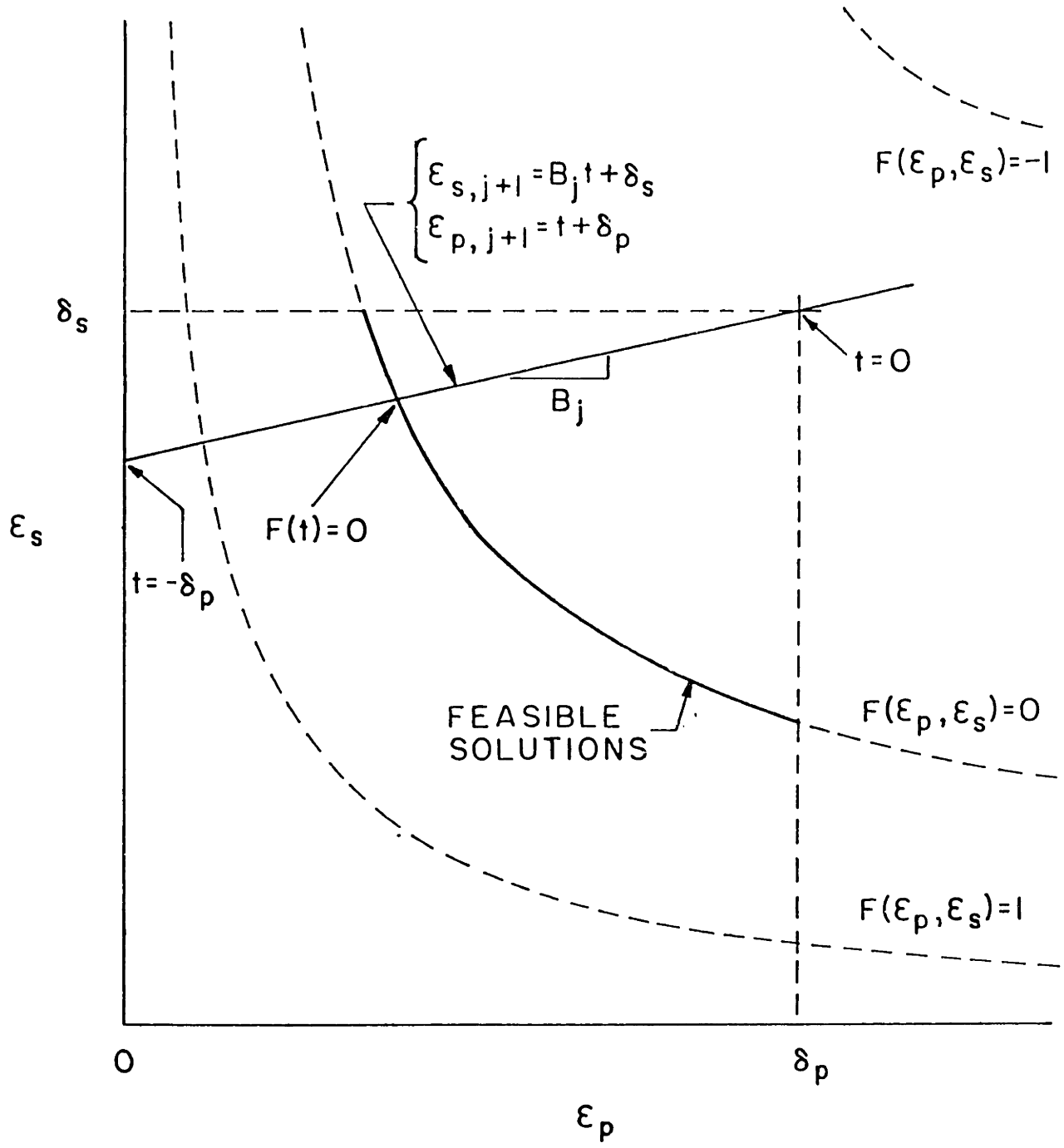


Fig. 6.8 Optimization problem illustrated in the $(\varepsilon_p, \varepsilon_s)$ parameter space.

($\varepsilon_s = \delta_s$, $\varepsilon_p = \delta_p$) which lies above the curve $F(\varepsilon_p, \varepsilon_s) = 0$, and on the line. A second point on the line, below the curve, can be obtained by choosing

$$t = t_b = \max\ (-\delta_p, -\delta_s/B_j),$$

in which case either $\varepsilon_p = 0$ or $\varepsilon_s = 0$ and $F(t_b) = +\infty$. Between these two points F(t) varies monotonically. The solution can now be obtained with a systematic trial-and-error procedure by choosing a new point $t_c = \frac{1}{2}\ (t_a + t_b)$ and evaluating $F(t_c)$. If $F(t_c) < 0$, assign $t_a = t_c$ and if $F(t_c) > 0$, assign $t_b = t_c$. The procedure is then repeated iteratively, halving the interval $[t_a, t_b]$ on each trial, until it converges. In practice, this procedure requires approximately 15 trials to reach an acceptable solution for which $-0.0001 \leq F(t) \leq 0$. The design parameters for the new filter trial can then be obtained from the solution t with the aid of the parametric equations (6.47a) and (6.47b).

Two filter examples were optimized by these procedures and the results are tabulated in Tables I and II. In the first example, the statistical word length was reduced by 1.74 bits. It was found by rounding of the coefficients that an actual improvement of 2 bits in the word length is obtained by using the statistically optimized design instead of the initial trial design. The second example resulted in a statistical word length reduction of 1.54 bits and an improvement of 1 bit in the actual word length. In both

TABLE I

TWO-PARAMETER OPTIMIZATION EXAMPLE A

FILTER PARAMETERS: $\delta_p = 0.03$, $\delta_s = 0.003$

$\Omega_p = 0.2\pi$, $\Omega_s = 0.25\pi$

FILTER STRUCTURE: CASCADE, $n = 6$

TRIAL	i =	1	2	3
OPTIMIZATION PARAMETERS	$\epsilon_{p,i}$	0.027038	0.020443	0.020618
	$\epsilon_{s,i}$	0.0025510	0.0029347	0.0029222
STATISTICAL WORD LENGTH (BITS) $y = 0.95$	W_p	12.78	11.01	11.04
	W_s	8.27	11.26	11.00
	W	12.78	11.26	11.04
SENSITIVITY RATIO	B_i	0.0066778	0.0081219	0.0080737
ACTUAL WORD LENGTH	W_a	12	11	10

STATISTICAL WORD LENGTH IMPROVEMENT: 1.74 BITS

ACTUAL WORD LENGTH IMPROVEMENT: 2 BITS

TABLE II

TWO-PARAMETER OPTIMIZATION EXAMPLE B

FILTER PARAMETERS: $\delta_p = 0.006$, $\delta_s = 0.01$

$\Omega_p = 0.1\pi$, $\Omega_s = 0.13\pi$

FILTER STRUCTURE: CASCADE, $n = 6$

TRIAL	$i =$	1	2	3
OPTIMIZATION PARAMETERS	$\epsilon_{p,i}$	0.0041751	0.0010926	0.0011100
	$\epsilon_{s,i}$	0.0050764	0.0099233	0.0098453
STATISTICAL WORD LENGTH (BITS) $y = 0.95$	W_p	14.90	13.29	13.30
	W_s	7.41	14.39	13.36
	W	14.90	14.39	13.36
SENSITIVITY RATIO	B_i	0.015067	0.033380	0.033065
ACTUAL WORD LENGTH	W_a	13	13	12

STATISTICAL WORD LENGTH IMPROVEMENT: 1.54 BITS

ACTUAL WORD LENGTH IMPROVEMENT: 1 BIT

examples, the optimization procedure converged to the final solution in only three trials on B_j.

6.4.2 Four-Parameter Statistical Approach

In the two-parameter approach, only the passband and stop band errors ε_p and ε_s are allowed to vary. This approach is useful especially for magnitude functions which are monotonic in the passband and stop band. For the case of equiripple functions, additional tradeoffs can often be made by shifting the passband and/or stop band edge frequencies, Ω_{pc} and Ω_{sc}, of the approximation function, into or out of the transition band. This, in effect, creates positive or negative "sensitivity guard bands." The direction of the shift is dependent on the sensitivity at the edge frequency relative to the in-band sensitivity. Often the sensitivity at the edge frequencies is greater than the in-band sensitivity, as can be seen in Fig. 6.3b, thus requiring positive "sensitivity guard bands" for word length minimization.

To demonstrate this approach, an equiripple passband and stop band lowpass filter magnitude function is assumed. For this case, there are four optimization parameters ε_p, ε_s, Ω_{pc}, and Ω_{sc}. An initial trial is chosen with parameters ($\varepsilon_{p,1}$, $\varepsilon_{s,1}$, $\Omega_{pc,1}$, $\Omega_{sc,1}$) and, as in the two-parameter approach, new trials are successively generated until the statistical word length is minimized. For each interation new parameters ($\varepsilon_{p,j+1}$, $\varepsilon_{s,j+1}$, $\Omega_{pc,j+1}$, $\Omega_{sc,j+1}$) must be determined for the

next trial.

As in the case of the two-parameter approach, the quantization step size necessary to meet the passband and stop band constraints can be derived as

$$Q_p = \frac{\sqrt{12}\ (\delta_p - \varepsilon_p)}{\frac{x}{2}\left[\max_{\omega'_{pu}} S(\omega'_{pu}) + \max_{\omega'_{p\ell}} S(\omega'_{p\ell})\right]} \tag{6.48}$$

and

$$Q_s = \frac{\sqrt{12}\ (\delta_s - \varepsilon_s)}{x \max_{\omega'_s} S(\omega'_s)} \ . \tag{6.49}$$

The essential difference between these equations and equations (6.23b) and (6.33) is that, in this case, the extreme frequencies ω'_{pu}, $\omega'_{p\ell}$, and ω'_s exclude the passband edge frequency Ω_p and the stop band edge frequency Ω_s. Two new functions, $Q_{\Omega p}$ and $Q_{\Omega s}$, are now defined as the quantization step sizes necessary to meet the constraints δ_p and δ_s, respectively, at the passband and stop band edge frequencies Ω_p and Ω_s.

The quantization step size $Q_{\Omega s}$ can be derived with the aid of Fig. 6.9. The j^{th} trial solution is designated as $|T_o|_j$ with the approximation edge frequency $\Omega_{sc,j}$. The next filter trial is designated as $|T_o|_{j+1}$ with the approximation edge frequency $\Omega_{sc,j+1}$ and error $\varepsilon_{s,j+1}$. It is assumed that the change in Ω_{sc}, given as

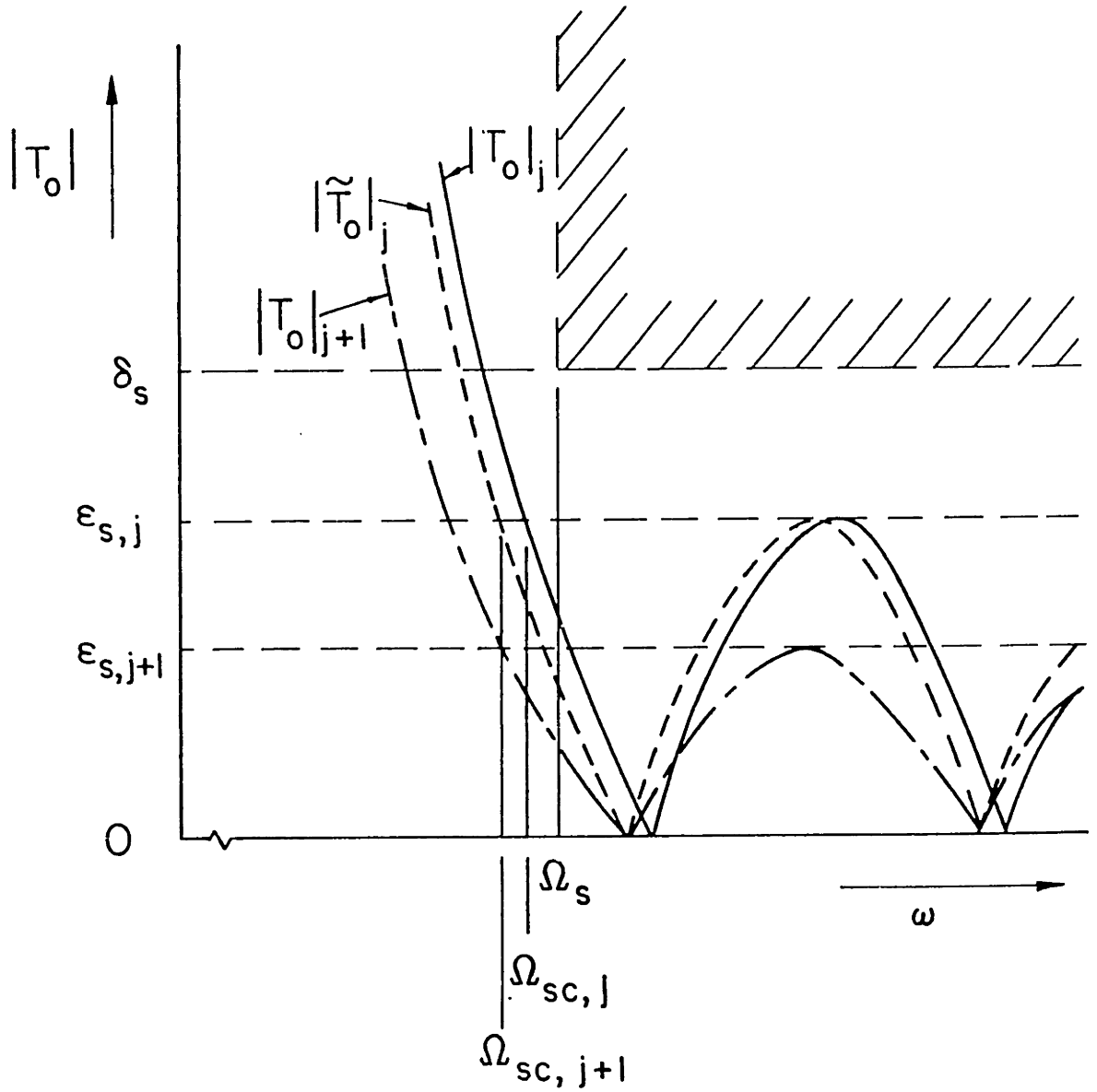


Fig. 6.9 Approximations at the stop band edge for determining $Q_{\Omega s}$.

$$\Delta\Omega_{sc} = \Omega_{sc,j+1} - \Omega_{sc,j}, \tag{6.50}$$

is sufficiently small so that when $\Omega_{sc,j}$ is changed by $\Delta\Omega_{sc}$, the resulting function, designated $|\tilde{T}_o|_j$, can be approximated as a shifted version of $|T_o|_j$ along the ω axis, in the region of the stop band edge Ω_s. The value of the new function $|\tilde{T}_o|_j$ at Ω_s can then be approximated using a first-order linear approximation of $|T_o|_j$, at Ω_s, as

$$|\tilde{T}_o(\Omega_s)|_j = |T_o(\Omega_s)|_j - C_j\Delta\Omega_{sc}, \tag{6.51a}$$

where

$$C_j \triangleq \left.\frac{\partial|T_o|_j}{\partial\omega}\right|_{\omega=\Omega_s}. \tag{6.51b}$$

In the new trial, $|T_o|_{j+1}$ represents not only the result of changing $\Omega_{sc,j}$ to $\Omega_{sc,j+1}$ but also of changing $\varepsilon_{s,j}$ to $\varepsilon_{s,j+1}$. This change will be approximated as a simple scaling of $|T_o|_j$ by the factor $\varepsilon_{s,j+1}/\varepsilon_{s,j}$. By combining the effects of shifting and scaling, the magnitude of the new trial $|T_o|_{j+1}$ at Ω_s can then be approximated as

$$|T_o(\Omega_s)|_{j+1} = \frac{\varepsilon_{s,j+1}}{\varepsilon_{s,j}}|\tilde{T}_o(\Omega_s)|_j = \frac{\varepsilon_{s,j+1}}{\varepsilon_{s,j}}\left(|T_o(\Omega_s)|_j - C_j(\Omega_{sc,j+1} - \Omega_{sc,j})\right). \tag{6.52}$$

With the aid of (6.15), $Q_{\Omega s}$ can now be given as

$$Q_{\Omega s} = \frac{\sqrt{12}\left[\delta_s - \frac{\varepsilon_{s,j+1}}{\varepsilon_{s,j}}\left(|T_o(\Omega_s)|_j - C_j(\Omega_{sc,j+1} - \Omega_{sc,j})\right)\right]}{xS(\Omega_s)} . \tag{6.53}$$

The approximations used in deriving $Q_{\Omega s}$ may not be accurate in the initial trials, but as successive filter trials converge to the final result, the change from trial to trial becomes smaller. As these changes become smaller, these first-order approximations become better, which in turn accelerates the convergence.

To determine $Q_{\Omega p}$, a similar set of approximations can be made. The use of the linear shift assumption gives

$$|\tilde{T}_o(\Omega_p)|_j = |T_o(\Omega_p)|_j - D_j(\Omega_{pc,j+1} - \Omega_{pc,j}), \tag{6.54a}$$

where

$$D_j \triangleq \left.\frac{\partial |T_o|_j}{\partial \omega}\right|_{\omega=\Omega_p} . \tag{6.54b}$$

The scaling assumption can also be made. For the passband, this scaling must be performed about unity instead of zero. This is accomplished as

$$|T_o(\Omega_p)|_{j+1} = 1 + \frac{\varepsilon_{p,j+1}}{\varepsilon_{p,j}} \left(|\tilde{T}_o(\Omega_p)|_j - 1 \right)$$

$$= 1 + \frac{\varepsilon_{p,j+1}}{\varepsilon_{p,j}} \left(|T_o(\Omega_p)|_j - D_j(\Omega_{pc,j+1} - \Omega_{pc,j}) - 1 \right) . \qquad (6.55)$$

The quantization step size $Q_{\Omega p}$ can then be derived, similar to that of (6.31), as

$$Q_{\Omega p} = \frac{\sqrt{12} \left[|T_o(\Omega_p)| - (K' - \delta_p) \right]}{xS(\Omega_p)}, \qquad (6.56)$$

where K' is defined as in (6.35) with the exception that ω_{pu} and $\omega_{p\ell}$ are replaced by ω'_{pu} and $\omega'_{p\ell}$, respectively. From (6.55), (6.56), and the new definition of K', $Q_{\Omega p}$ can then be expressed in the form

$$Q_{\Omega p} = \frac{\sqrt{12}}{xS(\Omega_p)} \left[\delta_p - (\delta_p - \varepsilon_{p,j+1}) E_j + \frac{\varepsilon_{p,j+1}}{\varepsilon_{p,j}} (|T_o(\Omega_p)|_j - 1) \right.$$

$$\left. - \frac{\varepsilon_{p,j+1}}{\varepsilon_{p,j}} (\Omega_{pc,j+1} - \Omega_{pc,j}) D_j \right], \qquad (6.57)$$

where

$$E_j \triangleq \frac{\max\limits_{\omega'_{pu}} S(\omega'_{pu}) - \max\limits_{\omega'_{p\ell}} S(\omega'_{p\ell})}{\max\limits_{\omega'_{pu}} S(\omega'_{pu}) + \max\limits_{\omega'_{p\ell}} S(\omega'_{p\ell})}. \tag{6.58}$$

The value E_j is determined from the filter realization for the j^{th} trial. As in the case of $Q_{\Omega s}$, the assumptions used to derive $Q_{\Omega p}$ become better as the trials converge to the solution.

The condition for statistical word length minimization can now be obtained by requiring

$$Q = Q_p = Q_s = Q_{\Omega p} = Q_{\Omega s}. \tag{6.59}$$

This gives three relations for determining the new parameter set ($\varepsilon_{p,j+1}$, $\varepsilon_{s,j+1}$, $\Omega_{pc,j+1}$, $\Omega_{sc,j+1}$). The fourth relation is again obtained from the intrinsic function

$$F(\varepsilon_p, \varepsilon_s, \Omega_{pc}, \Omega_{sc}) = 0. \tag{6.60}$$

Following the same approach as the two-parameter problem, the parameters can be written in parametric form. Using (6.48), (6.49), (6.53), (6.57), (6.58), and (6.59) they become

$$\varepsilon_{s,j+1} = \varepsilon_s = B_j t + \delta_s, \tag{6.61a}$$

$$\varepsilon_{p,j+1} = \varepsilon_p = t + \delta_p, \tag{6.61b}$$

$$\Omega_{sc,j+1} = \Omega_{sc} = \Omega_{sc,j} + \frac{1}{C_j}\left[|T_o(\Omega_s)|_j - \frac{\varepsilon_{s,j}}{B_j t + \delta_s}\left(\delta_s + \frac{S(\Omega_s)\, B_j t}{\max\limits_{\omega'_s} S(\omega'_s)}\right)\right], \tag{6.61c}$$

$$\Omega_{pc,j+1} = \Omega_{pc} = \Omega_{pc,j} + \frac{1}{D_j}\left[|T_o(\Omega_p)|_j - 1 + \frac{\varepsilon_{p,j}}{t+\delta_p}(\delta_p + G_j t)\right], \tag{6.61d}$$

where

$$G_j = \frac{S(\Omega_p) + \frac{1}{2}\left[\max\limits_{\omega'_{pu}} S(\omega'_{pu}) - \max\limits_{\omega'_{p\ell}} S(\omega'_{p\ell})\right]}{\frac{1}{2}\left[\max\limits_{\omega'_{pu}} S(\omega'_{pu}) + \max\limits_{\omega'_{p\ell}} S(\omega'_{p\ell})\right]}, \tag{6.62}$$

and

$$B_j = \frac{\max\limits_{\omega'_s} S(\omega'_s)}{\frac{1}{2}\left[\max\limits_{\omega'_{pu}} S(\omega'_{pu}) + \max\limits_{\omega'_{p\ell}} S(\omega'_{p\ell})\right]}. \tag{6.63}$$

The values G_j and B_j are determined from the filter

realization for the j^{th} trial. Upon substitution in (6.60), the problem again reduces to one of finding a zero of a non-linear function F(t) in the single variable t. The same algorithm that was described for the two-parameter approach applies here.

Examples

To demonstrate the four-parameter approach the elliptic approximation was again chosen. The intrinsic function for (6.60) can be written in the same form as (6.44). Expressions for determining the maximum error frequencies ω'_{pu}, $\omega'_{p\ell}$, and ω'_s are given in Section 6.5.

Two examples were analyzed with the four-parameter approach and the results are summarized in Tables III and IV. The filter specifications for the examples are the same as those used in the two-parameter optimization examples in Tables I and II. In comparison, slightly greater improvement in the statistical word length was obtained from the four-parameter approach. As can be seen from the examples, this does not necessarily guarantee that the actual word length will be improved by using the four-parameter approach instead of the two-parameter approach. In Tables III and IV the growth of the "sensitivity guard bands" is evident as the new trials are generated.

TABLE III

FOUR-PARAMETER OPTIMIZATION EXAMPLE A

FILTER PARAMETERS:	$\delta_p = 0.03$,	$\delta_s = 0.003$		
	$\Omega_p = 0.2\pi$,	$\Omega_s = 0.25\pi$		
FILTER STRUCTURE:	CASCADE,	$n = 6$		
TRIAL	$i =$	1	2	3
OPTIMIZATION PARAMETERS	$\varepsilon_{p,i}$	0.027038	0.019967	0.020002
	$\varepsilon_{s,i}$	0.0025510	0.0029745	0.0029723
	$\Omega_{pc,i}$	0.2π	0.199985π	0.199976π
	$\Omega_{sc,i}$	0.25π	0.249943π	0.249930π
STATISTICAL WORD LENGTH (BITS) $y = 0.95$	W_p	12.78	10.93	10.94
	W_s	6.69	11.05	10.93
	$W_{\Omega p}$	12.73	10.90	10.94
	$W_{\Omega s}$	8.27	11.17	10.93
	W	12.78	11.17	10.94
ACTUAL WORD LENGTH	W_a	12	11	11
STATISTICAL WORD LENGTH IMPROVEMENT: 1.84 BITS				
ACTUAL WORD LENGTH IMPROVEMENT: 1 BIT				

TABLE IV

FOUR-PARAMETER OPTIMIZATION EXAMPLE B

FILTER PARAMETERS: $\delta_p = 0.006$, $\delta_s = 0.01$

$\Omega_p = 0.1\pi$ $\Omega_s = 0.13\pi$

FILTER STRUCTURE: CASCADE, $n = 6$

TRIAL i =		1	2	3	4
OPTIMIZATION PARAMETERS	ε_p	0.0041750	0.0010925	0.0011533	0.0011601
	ε_s	0.0050764	0.0099848	0.0099479	0.0099429
	Ω_{pc}	0.1π	0.100055π	0.100284π	0.100309π
	Ω_{sc}	0.13π	0.129989π	0.129974π	0.129974π
STATISTICAL WORD LENGTH (BITS) $y = 0.95$	W_p	14.88	13.19	13.22	13.22
	W_s	5.81	15.13	13.35	13.22
	$W_{\Omega p}$	14.92	13.37	13.23	13.22
	$W_{\Omega s}$	7.41	14.62	13.25	13.22
	W	14.92	15.13	13.35	13.22
ACTUAL WORD LENGTH	W_a	13	13	13	10

STATISTICAL WORD LENGTH IMPROVEMENT: 1.70 BITS

ACTUAL WORD LENGTH IMPROVEMENT: 3 BITS

Bandpass Example

To demonstrate how the statistical method might be applied to the case of bandpass filters, the bandpass example used by Avenhaus [57] was chosen (see Fig. 62a). The filter approximation was synthesized from a lowpass, fourth-order, elliptic prototype, using the lowpass to bandpass transformation [61],

$$g(z^{-1}) = \frac{z^{-1}(z^{-1}-\alpha)}{1 - \alpha z^{-1}} . \qquad (6.64)$$

Optimization was performed on the lowpass prototype with the appropriate sensitivities taken from the bandpass implementation. The mapping parameter, α, in (6.64) was fixed to the same value for all of the trials. The results are given in Tables V and VI for the two- and four-parameter approaches, respectively. The statistical word length was minimized to 10.26 bits for the two-parameter approach and to 10.11 bits for the four-parameter approach. In both cases, the actual coefficient word length was improved by one bit for coefficient rounding.

6.5 CALCULATION OF THE MAXIMUM ERROR FREQUENCIES FOR THE ELLIPTIC APPROXIMATION

The frequencies at which $|T_o|$ has its maximum error for the lowpass elliptic approximation can be determined from the elliptic functions [59]. Expressions for evaluating the

TABLE V

TWO-PARAMETER BANDPASS EXAMPLE C

FILTER PARAMETERS:	$\delta_p = 0.03$,	$\delta_s = 0.01$		
(FOR LP PROTOTYPE)	$\Omega_p = 0.0555556$			
	$\Omega_s = 0.1097430$			
MAPPING PARAMETER:	$a = 0.19153785$			
FILTER STRUCTURE:	CASCADE,	$n = 8$ (BP)		
TRIAL	$i =$	1	2	3
OPTIMIZATION PARAMETERS	$\varepsilon_{p,i}$	0.015781	0.0045106	0.0047290
	$\varepsilon_{s,i}$	0.0052606	0.0098419	0.0096119
STATISTICAL WORD LENGTH (BITS) $y = 0.95$	W_p	11.49	10.23	10.26
	W_s	5.73	11.54	10.21
	W	11.49	11.54	10.26
SENSITIVITY RATIO	B_i	0.0061316	0.015329	0.014818
ACTUAL WORD LENGTH	W_a	10	10	9
STATISTICAL WORD LENGTH IMPROVEMENT: 1.23 BITS				
ACTUAL WORD LENGTH IMPROVEMENT: 1 BIT				

TABLE VI

FOUR-PARAMETER BANDPASS EXAMPLE C

FILTER PARAMETERS:	$\delta_p = 0.03$,	$\delta_s = 0.01$		
(FOR LP PROTOTYPE)	$\Omega_p = 0.0555556$			
	$\Omega_s = 0.1097430$			
MAPPING PARAMETER:	$\alpha = 0.19153785$			
FILTER STRUCTURE:	CASCADE,	$n = 8$ (BP)		
TRIAL	$i =$	1	2	3
OPTIMIZATION PARAMETERS (FOR LP PROTOTYPE)	$\varepsilon_{p,i}$	0.015781	0.0058399	0.0062124
	$\varepsilon_{s,i}$	0.0052606	0.0099319	0.0098604
	$\Omega_{pc,i}$	0.0555556π	0.0571799π	0.0574176π
	$\Omega_{sc,i}$	0.1097430π	0.1096700π	0.1095758π
STATISTICAL WORD LENGTH (BITS) $y = 0.95$	W_p	11.17	10.07	10.11
	W_s	4.10	11.16	10.11
	$W_{\Omega p}$	11.89	10.15	10.11
	$W_{\Omega s}$	5.73	11.26	10.11
	W	11.89	11.26	10.11
ACTUAL WORD LENGTH	W_a	10	11	9
STATISTICAL WORD LENGTH IMPROVEMENT: 1.78 BITS				
ACTUAL WORD LENGTH IMPROVEMENT: 1 BIT				

frequencies for the two- and four-parameter optimization are given. It is assumed that the z plane elliptic approximations are obtained from s plane designs by the bilinear transformation [1], [2], [3].

For the two-parameter approach, the sets of frequencies ω_{pu}, $\omega_{p\ell}$, and ω_s are given as

$$\omega_{pu} = \begin{cases} \{0, \omega_{pui}\} & n \text{ odd} \\ \{\omega_{pui}\} & n \text{ even} \end{cases} \tag{6.65}$$

$$\omega_{p\ell} = \begin{cases} \{\omega_{p\ell i}, \Omega_p\} & n \text{ odd} \\ \{0, \omega_{p\ell i}, \Omega_p\} & n \text{ even} \end{cases} \tag{6.66}$$

and

$$\omega_s = \begin{cases} \{\Omega_s, \omega_{si}\} & n \text{ odd} \\ \{\Omega_s, \omega_{si}, \pi\} & n \text{ even} \end{cases} \tag{6.67}$$

where the sets of frequencies ω_{pui}, $\omega_{p\ell i}$, and ω_{si} are given in (6.71), (6.72), and (6.73), respectively.

For the four-parameter approach, ω'_{pu}, $\omega'_{p\ell}$, and ω'_s are given as

$$\omega'_{pu} = \begin{cases} \{0, \omega_{pui}\} & n \text{ odd} \\ \{\omega_{pui}\} & n \text{ even} \end{cases} \tag{6.68}$$

$$\omega'_{p\ell} = \begin{cases} \{\omega_{p\ell i}\} & n \text{ odd} \\ \{0,\ \omega_{p\ell i}\} & n \text{ even} \end{cases} \tag{6.69}$$

and

$$\omega'_{s} = \begin{cases} \{\omega_{si}\} & n \text{ odd} \\ \{\omega_{si},\ \pi\} & n \text{ even} \end{cases} \tag{6.70}$$

The sets of frequencies ω_{pui}, $\omega_{p\ell i}$, and ω_{si} are given as

$$\omega_{pui} = 2 \tan^{-1} \left[\tan \left(\frac{\Omega_{pc}}{2} \right) \text{Sn} \left(\frac{(2i-1+\nu)\ K(k)}{n},\ k \right) \right]$$

$$\text{for } i = 1,2,3,\ldots \frac{n-\nu}{2}, \tag{6.71}$$

$$\omega_{p\ell i} = 2 \tan^{-1} \left[\tan \left(\frac{\Omega_{pc}}{2} \right) \text{Sn} \left(\frac{(2i-\nu)\ K(k)}{n},\ k \right) \right]$$

$$\text{for } i = 1,2,3,\ldots \frac{n-2+\nu}{2}, \tag{6.72}$$

and

$$\omega_{si} = 2 \tan^{-1} \left[\frac{\tan \left(\frac{\Omega_{sc}}{2} \right)}{\text{Sn} \left(\frac{(2i-\nu)\ K(k)}{n},\ k \right)} \right]$$

$$\text{for } i = 1,2,\ldots \frac{n-2+\nu}{2}, \tag{6.73}$$

where

Sn(u,k) = the elliptic sine function,

K(k) = the complete elliptic integral of the first kind,

n = the order of the filter,

k = the selectivity factor (see (6.45)),

and

$$\nu = \begin{cases} 1 & n \text{ odd} \\ 0 & n \text{ even} \end{cases}$$

An algorithm for computing K(k) is given by Hastings [62] and algorithms for computing Sn(u,k) are given by Gold and Rader [2] and by Byrd and Friedman [63]. These computations are not as formidable as they first appear. The entire optimization procedure could easily be performed on a programmable desk calculator.

6.6 CONCLUSIONS AND ADDITIONAL COMMENTS

From the examples in Tables I-VI it can be seen that minimizing the statistical word length usually leads to a reduction in the actual coefficient word length by approximately one to three bits for coefficient rounding. There is no assurance, however, that the actual global minimum word length will be obtained. This can be seen in the bandpass example, where an improvement of one bit in the actual coefficient word length was obtained with the statistical

approach, whereas a three-bit improvement for the same example was found by Avenhaus [57] who used a nonlinear integer programming approach.

As the statistical optimization procedure is very efficient computationally, we conjecture that this approach might be useful as an initial phase in a nonlinear integer programming approach if a further reduction of the coefficient word length is desired. The statistical approach could be used first to determine the global minimum statistical word length. This solution could then serve as an initial starting point for a nonlinear, integer, local search procedure to determine the actual minimum word length. This two-phase procedure would, it is hoped, result in a considerable reduction in the total computation time.

For the case of algorithmic synthesis techniques, it was proposed that the implied intrinsic function $F(\varepsilon_p, \varepsilon_s)$ be determined numerically over the range $\varepsilon_p \leq \delta_p$, $\varepsilon_s \leq \delta_s$. A more efficient technique might be to attempt to incorporate the sensitivity relations more directly into the algorithmic synthesis procedure. This concept might lead to some interesting possibilities. It is expected that the feasibility of such an approach would strongly depend on the type of algorithm used.

The statistical optimization procedure proposed in this paper should also be applicable, with slight modifications,

to the problem of sensitivity minimization or tolerance maximization in analog filters. As the sensitivity problem for analog filters is truly a statistical one, this method might be even more suited to this problem than to the digital case. A savings of 1 to 2 bits in the digital case would correspond to factors of 2 to 4 in tolerance improvement in the analog case.

Chapter VII

A COMPARISON OF DIGITAL FILTER STRUCTURES ON THE BASIS OF COEFFICIENT WORD LENGTH

7.1 INTRODUCTION

In this chapter, we will compare a number of different classes of recursive filter structures which have been proposed by numerous authors in the recent literature. The comparison is made primarily on the basis of coefficient word length and on the number of multiplies and adds required for the various structures. We will also attempt to make comments and comparisons with respect to issues such as the range of coefficient values, modularity of the structures, complexity of the structures, parallelism, and other properties which affect the design of these structures and their realizations in terms of hardware. The analyses of these structures were performed with the aid of the general purpose computer-aided digital network analysis package (CADNAP) which was discussed in Chapter 5.

For the comparison, we chose the eighth-order elliptic bandpass filter example used in Chapter 6. The filter specifications are:

$$\varepsilon_p = 0.0157816, \qquad \varepsilon_s = 0.00526063,$$
$$\Omega_{p1} = 0.4111111\ \pi, \qquad \Omega_{s1} = 0.3847015\ \pi,$$
$$\Omega_{p2} = 0.4666667\ \pi, \qquad \Omega_{s2} = 0.4944444\ \pi.$$

This example was chosen for the reason that the elliptic filters are a widely used class of recursive filters and because it enabled us to show that for certain classes of filter designs, such as the elliptic designs and the bandpass transformed designs, we can often reduce the number of multiplies required (in some classes of structures) by taking advantage of the properties of these designs.

From the comparisons, based on this example, we can draw some conclusions as to what types of structures might be good candidates for practical and efficient implementation of digital filters. However, discretion must be used in extending these results to other similar types of structures or to other filter examples. Clearly, the problem of comparing filter structures is an open-ended problem and, at best, we can only expect to arrive at limited conclusions and develop an "intuitive feeling" for what types of structures are most useful in a particular situation.

7.2 THE FILTER STRUCTURE EXAMPLES

Thirteen different recursive filter structures were chosen for comparison. Although these structures do not comprise all of the possible known structures, they were chosen to represent a typical cross section of the various methods and philosophies for recursive filter structure synthesis known by this author to date. It is hoped that this comparison will provide further insight into which methods and

philosophies look promising for further research and that it will stimulate investigation in these areas.

The various structures and any unique properties of these structures will be discussed in this section. The comparisons between them will then be given in table form and discussed in Section 7.3. No attempt will be made to explain the details for synthesis of each of the different types of structures, as these details are readily available in the references. Equations expressing the system functions for the various types of structures (for fourth-order system functions) in terms of their coefficients, except the ladder structures, can be found in Chapter 4.

7.2.1 Common Filter Structures

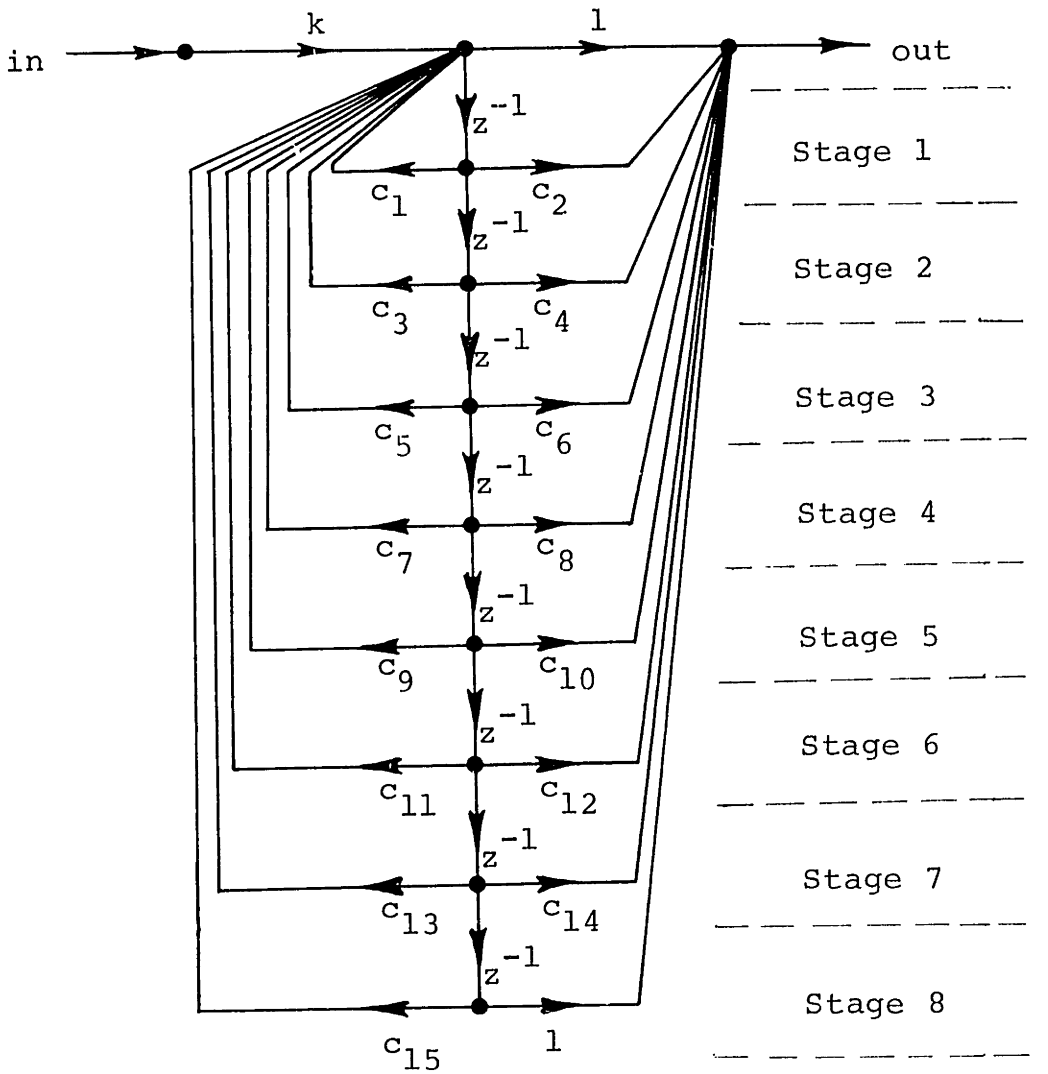
The first three types of structures which were examined are the well-known direct form II structure (sometimes referred to as the canonic form), the cascade form structure, and the parallel form structure, where the cascade and parallel forms are implemented with first- and/or second-order direct form II sections. These three classes of structures, in various forms, were perhaps the first types of structures to be considered in digital signal processing. Their properties have been studied extensively and are well-known [1], [2], [3], [4].

The direct form structures are synthesized from the ratio of polynomials comprising the system function and the

coefficients in the structure take the values of the coefficients in these polynomials. The direct form II structure for the eight-order bandpass filter example is given in Fig. 7.1.

It was domonstrated by Kaiser [1] that the direct form recursive structures are highly sensitive to variations in the coefficients, especially in situations where the poles of the system function are tightly clustered in the z plane. He demonstrated this by showing that the accuracy requirements on the coefficients of the numerator and denominator polynomials of the system function become increasingly more severe as the order of the polynomials increases and as the roots of the polynomials become closely spaced. As the coefficients of the direct form structure correspond to the coefficients of the polynomials, the same sensitivity properties apply to the structure. Thus, we would expect the direct form structures to require relatively high coefficient word lengths.

In the parallel form structures and the cascade form structures, the above problems of large order polynomials are avoided by implementing the filter in terms of parallel and cascade combinations of low order filter sections, respectively. Therefore, we can expect smaller coefficient word lengths for these classes of structures. The parallel form structures are synthesized from a partial-fraction expansion of the system function and the cascade form structures are synthesized by factoring the numerator and denominator polynomials of the system function into first- and/or second-order polynomials.



$k = 0.005656462366$

$c_1 = 1.479757145$ $\quad c_2 = -1.387818861$

$c_3 = -4.548129731$ $\quad c_4 = 3.990834175$

$c_5 = 4.352241828$ $\quad c_6 = -3.801090558$

$c_7 = -6.802822262$ $\quad c_8 = 5.978059098$

$c_9 = 4.094876357$ $\quad c_{10} = -3.801090558$

$c_{11} = -4.026604318$ $\quad c_{12} = 3.990834175$

$c_{13} = -0.7833447265$ $\quad c_{14} = -1.387818861$

$c_{15} = 1.232007442$

Fig. 7.1 Direct form II structure, (Example 1).

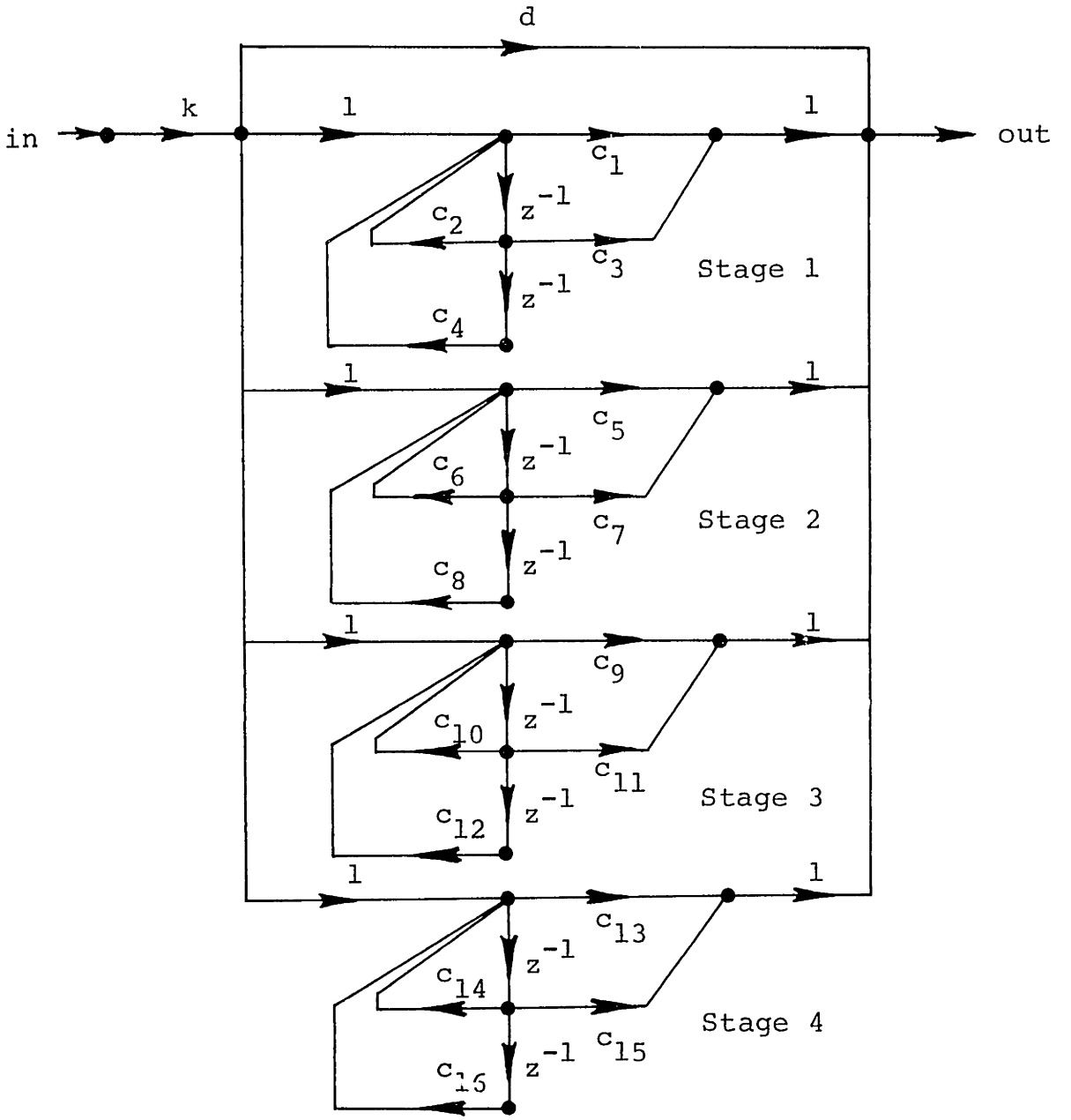
These structures are given, for the eight-order bandpass example, in Figures 7.2 and 7.3, respectively.

A useful property of the cascade structure for the case of system functions where zeros are located on the unit circle (e.g. Butterworth, Tchebyshev, and elliptic designs) is that some of the coefficients in the structure are unity and do not have to be implemented as hardware multiplies. This results in a savings of one multiply per stage in the cascade structure.

7.2.2 Cascade Structures

As we have observed in Chapter 4, in addition to the direct form II sections for the cascade structures, many other network configurations for implementation of these sections are possible and have been proposed by various authors [2], [3], [32], [33], [34]. Three different forms of second-order sections were selected from these possibilities for the implementation of the eight-order bandpass example. They are the coupled form sections [2], [3] and the network configurations B and F proposed by Avenhaus [34].

The coupled form, second-order sections are of interest because the coefficients can be chosen in a manner such that the allowed pole positions, upon quantization of the coefficients in fixed-point arithmetic, lie on a rectangular grid in the z plane. For the eight-order bandpass example, they were chosen in this way and the resulting structure is given in



k =	0.10000	d =	0.07220910762
c_1 =	0.2034508401	c_9 =	-0.2052671178
c_2 =	0.2855823838	c_{10} =	0.196901665
c_3 =	0.9026252904	c_{11} =	-0.2413054652
c_4 =	-0.9116860616	c_{12} =	-0.9695353354
c_5 =	0.182701196	c_{13} =	-0.1965294023
c_6 =	0.4457342317	c_{14} =	0.5515388641
c_7 =	-0.9457352763	c_{15} =	0.298888793
c_8 =	-0.913078673	c_{16} =	-0.9705899009

Fig. 7.2 Parallel form structure, (Example 2).

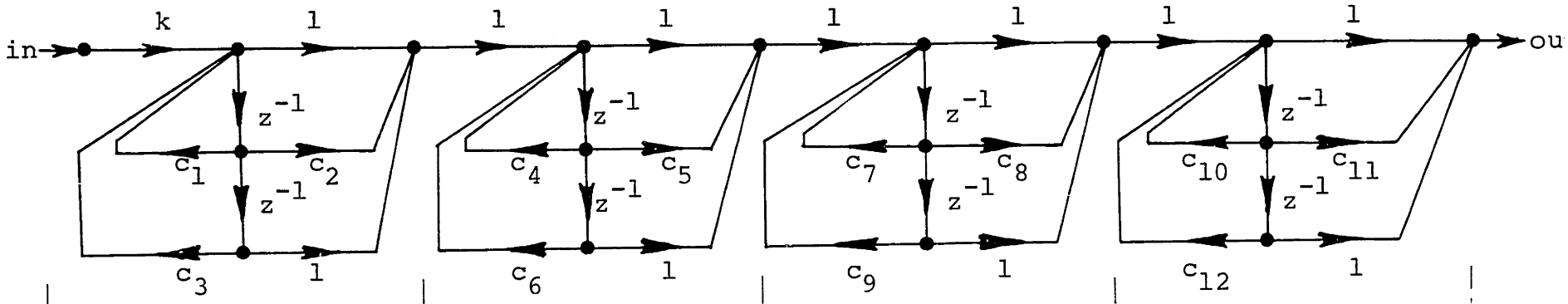
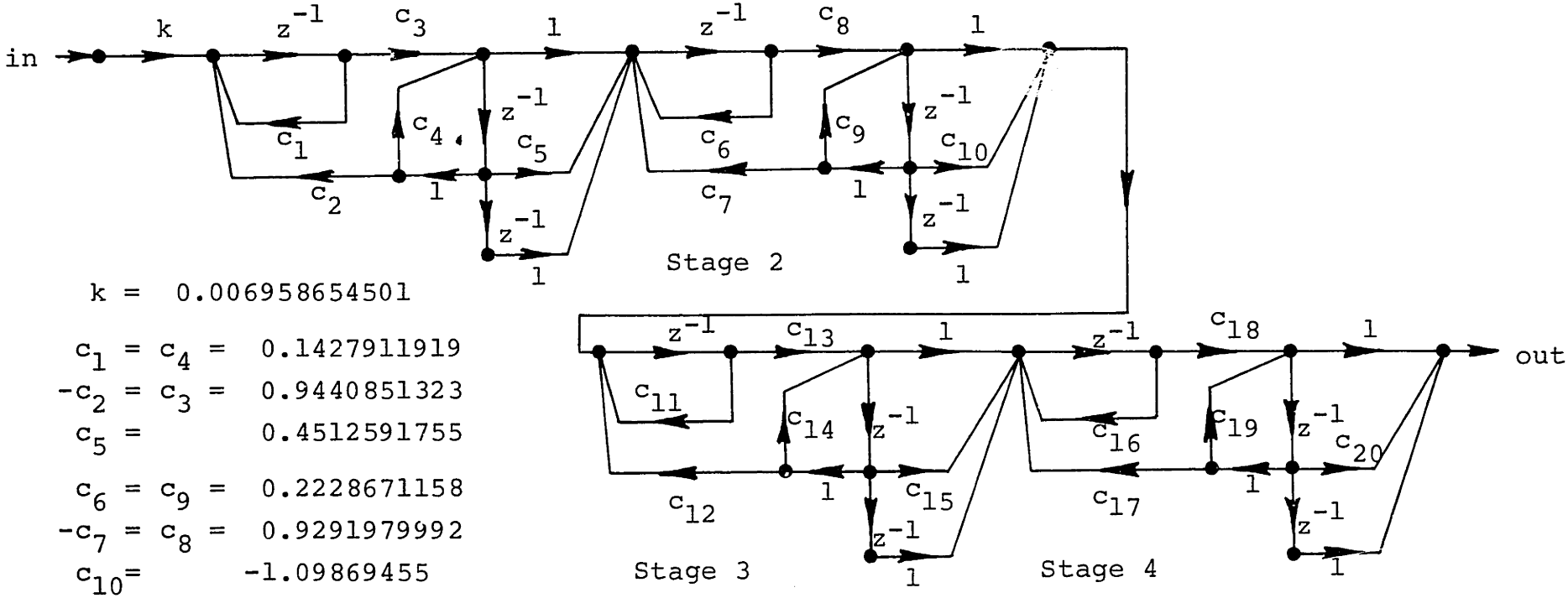


Fig. 7.3 Cascade form (direct form II sections) structure, (Example 3).

Fig. 7.4. Two undesirable properties of this structure are that it is not canonical with respect to the number of multiplies and that an additional one-sample delay per cascade stage is introduced in the filter.

In the design of second-order filter sections for the synthesis of cascade form structures, a very useful technique for obtaining qualitative information about the sensitivity properties of second-order sections is to plot the allowed positions that the poles and zeros can take in the z plane when the coefficients in the structure are quantized to finite word lengths. By observing where the most densely spaced positions are in the z plane, we can obtain an idea of the types of designs for which each of these different second-order sections are most suitable. Unfortunately, the technique is not very applicable to sections which are greater than second-order. An extensive analysis of numerous second-order filter sections for cascade structures using this technique has been made by Avenhaus [34]. Two of these configurations which appeared to be of particular interest for our bandpass example, other than the direct form II and coupled form sections, are the configurations B and F (see Avenhaus [34]). These structures for the bandpass example are given in Figures 7.5 and 7.6, respectively. The second network configuration type F is of special interest as it was noted by Avenhaus that the density of allowed pole positions was relatively high near the unit circle and, therefore, it should be particularly



k = 0.006958654501

$c_1 = c_4 =$ 0.1427911919

$-c_2 = c_3 =$ 0.9440851323

$c_5 =$ 0.4512591755

$c_6 = c_9 =$ 0.2228671158

$-c_7 = c_8 =$ 0.9291979992

$c_{10} =$ -1.09869455

$c_{11} = c_{14} =$ 0.0984508325

$-c_{12} = c_{13} =$ 0.9797156572

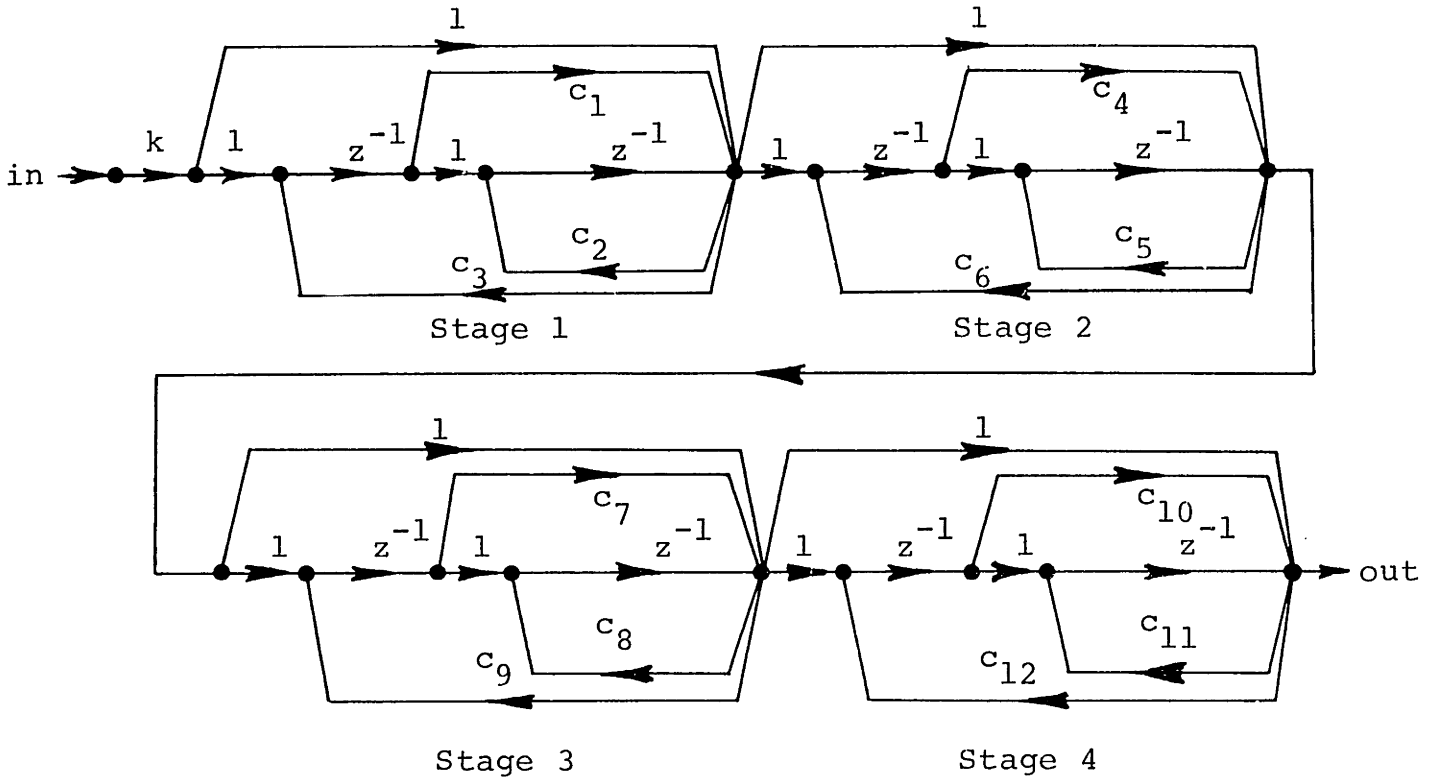
$c_{15} =$ -0.0099665157

$c_{16} = c_{19} =$ 0.275769432

$-c_{17} = c_{18} =$ 0.9458018404

$c_{20} =$ -0.7304169706

Fig. 7.4 Cascade (Coupled form) structure, (Example 4).



$k = 0.005656462366$

$c_1 = 0.4512591755$ $\quad c_7 = -0.0099665157$

$c_2 = 0.6969890843$ $\quad c_8 = 0.1872387759$

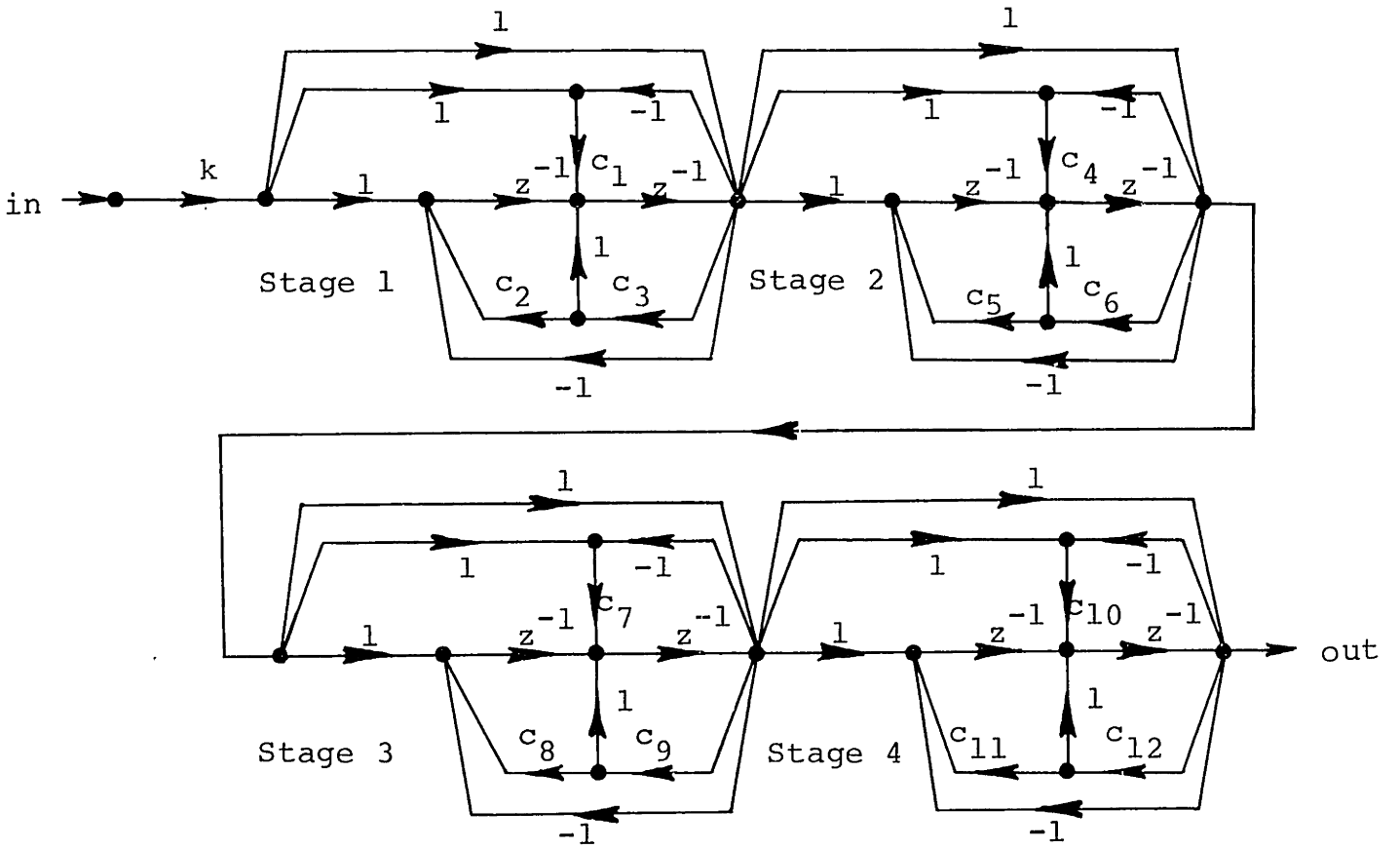
$c_3 = -0.9116860616$ $\quad c_9 = -0.9695353354$

$c_4 = -1.09869455$ $\quad c_{10} = -0.7304169706$

$c_5 = -0.5574603304$ $\quad c_{11} = -0.157396471$

$c_6 = -0.913078673$ $\quad c_{12} = -0.9705899009$

Fig. 7.5 Cascade structure with Avenhaus type B sections, (Example 5).



k = 0.005656462366

c_1 =	0.4512591755	c_7 =	-0.0099665157
c_2 =	0.1198547195	c_8 =	0.1629691618
c_3 =	0.7368415593	c_9 =	0.1869351493
c_4 =	-1.09869455	c_{10} =	-0.7304169706
c_5 =	-0.1331188504	c_{11} =	-0.1644141906
c_6 =	-0.6529603187	c_{12} =	-0.1788781065

Fig. 7.6 Cascade structure with Avenhaus type F sections (Example 6).

suited for recursive bandpass filter designs. Both types of configurations also take advantage of the fact that when the zeros of the system function are on the unit circle, one coefficient per second-order section is unity and does not have to be implemented as a hardware multiply.

7.2.3 Structure Synthesis Using Bandpass Transformations

In the synthesis of bandpass filters, such as the example that we are considering, we can first synthesize a lowpass prototype of the filter and then transform this function to a bandpass function [61]. In this transformation we replace all functions z^{-1} by the allpass mapping, $g(z^{-1})$, as

$$z^{-1} \xrightarrow[\text{by}]{\text{replace}} g(z^{-1}) = - \frac{z^{-1}(z^{-1}-\alpha)}{1 - \alpha z^{-1}} , \qquad (7.1)$$

where α is the mapping parameter. This same process can be incorporated into the synthesis of filter structures by first synthesizing an appropriate lowpass filter and then replacing all delays in the structure by a circuit which realizes the mapping in (7.1). A circuit for doing this is given in Fig. 7.7. The parameter α determines the bandpass center frequency ω_o to which the DC value of the lowpass filter function is mapped. The DC value of the lowpass filter corresponds to

$$g(z^{-1}) = e^{j0} = 1.$$

Therefore, to obtain the center frequency of the bandpass

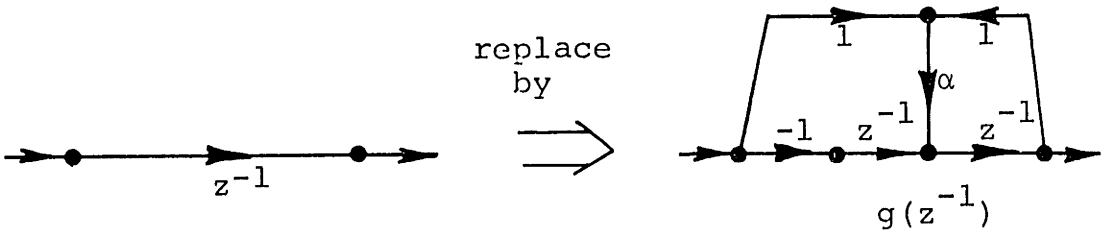


Fig. 7.7 Lowpass to bandpass mapping.

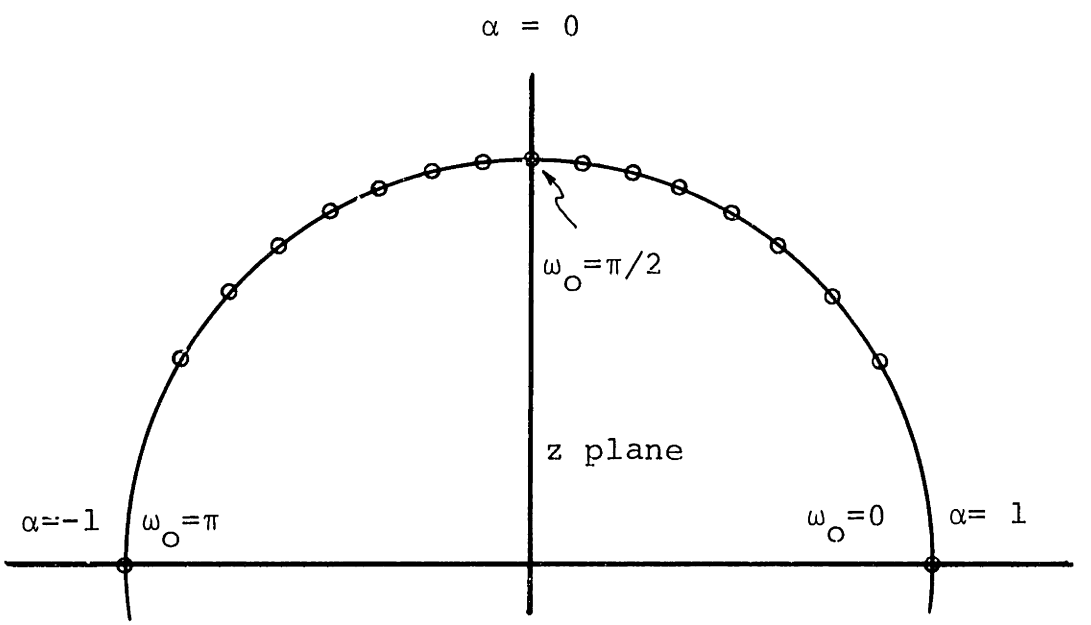


Fig. 7.8 Allowed center frequency locations of the lowpass to bandpass mapping for α quantized to 3 bits fixed-point (quantization step size = 0.125).

filter we must solve the expression

$$e^{j0} = 1 = - \frac{z_o^{-1}(z_o^{-1}-\alpha)}{1 - \alpha z_o^{-1}} \tag{7.2}$$

for z_o. Rearranging terms gives

$$z_o^2 - 2\alpha z_o + 1 = 0$$

and therefore,

$$z_o = \alpha \pm \sqrt{\alpha^2 - 1} \; . \tag{7.3}$$

For bandpass mappings we require

$$-1 \leq \alpha \leq 1 \tag{7.4}$$

and, therefore, the radical term in (7.3) corresponds to the imaginary part of z_o and

$$|z_o| = |e^{j\omega_o}| = 1 \tag{7.5}$$

The bandpass center frequency ω_o can then by expressed in terms of α as

$$\cos \omega_o = \alpha. \tag{7.6}$$

For quantized values of α we can determine the allowable center frequencies ω_o. These frequencies are plotted in Fig. 7.8 for α quantized in increments of 2^{-3} from -1 to 1. We can

observe from this figure that bandpass mappings can be performed most accurately when the center frequencies of the bandpass filter are near $\pi/2$.

We can also observe from the first-order sensitivity relation derived in Chapter 3 (Section 3.5) that all of the sensitivity properties of the lowpass filter are mapped in frequency to the bandpass case. Thus, if we have a structure which has good sensitivity properties for lowpass filter designs, we can also use it to design bandpass filters with the same low sensitivity properties using the mapping technique.

An additional attractive feature of the mapping approach for the design of bandpass filters is that it allows us to reduce the number of multiplies required in the structures. For example, if we consider an arbitrary system function with eight poles and eight zeros (in conjugate pairs), there are 17 degrees of freedom to be considered, two for each pole pair, two for each zero pair, and one for the gain constant. Thus, the minimum number of multiplies that we need to perform in a structure to realize this system function is 17. If we constrain the eight zeros to be on the unit circle, then we only have one degree of freedom per zero pair and the total number of degrees of freedom is 13. For this case, the minimum number of multiplies that we need to perform in a structure is 13 (e.g. the cascade structures in Figures 7.3, 7.5, and 7.6). If we make the further restriction that the

eighth-order system function is obtained by mapping a fourth-order system function (with zeros on the unit circle) according to (7.1), then we require only 11 degrees of freedom. Two degrees of freedom are necessary for each pole pair and one for each zero pair of the fourth-order system function. In addition, we need one degree of freedom for each delay in the fourth-order lowpass prototype, which is mapped according to (7.1), and one degree of freedom for the gain constant. Thus, we should only require a minimum of 11 multiplies in the structure. This can be achieved using the mapping approach for the synthesis of the structure. We will, therefore, consider a structure for this eighth-order bandpass example to be truly canonical if it only requires 11 multiplies.

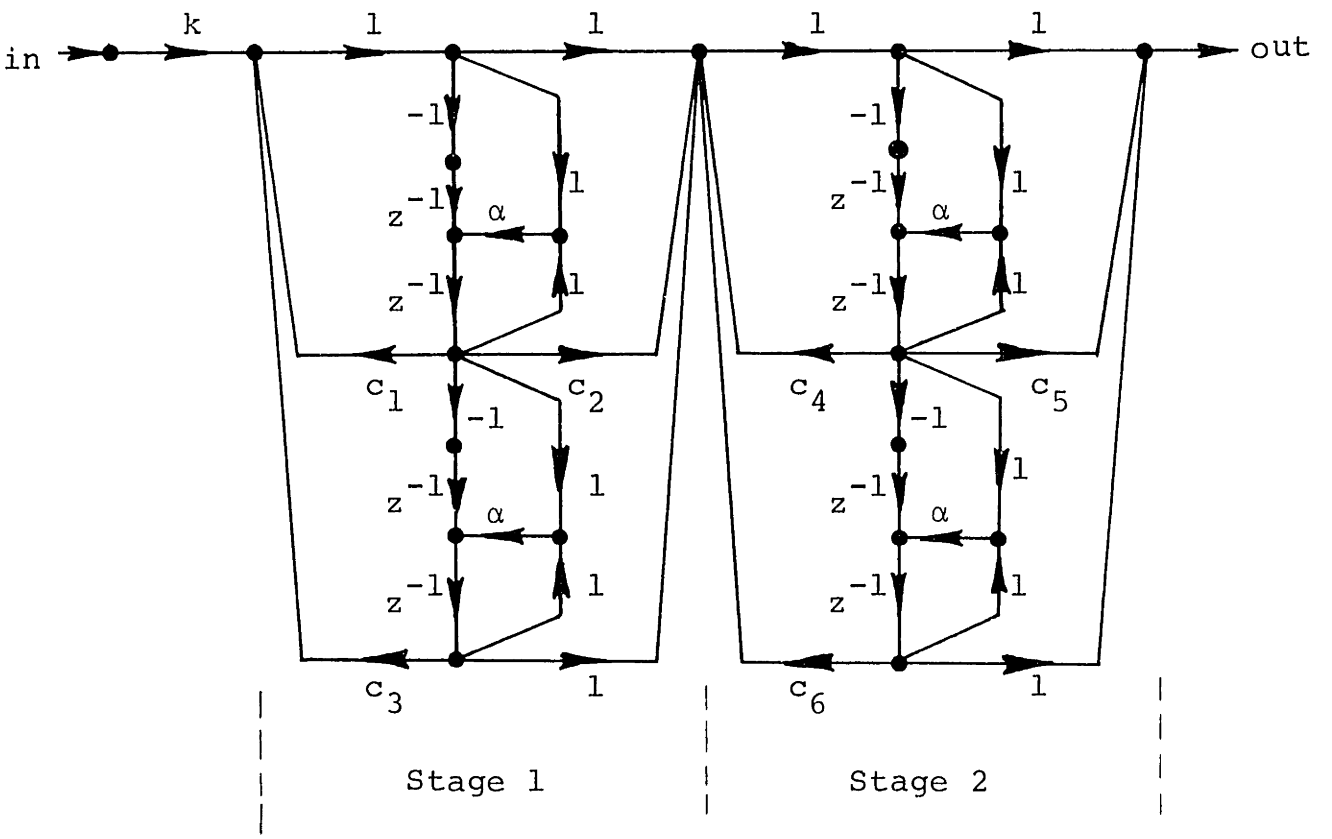
Two examples were chosen for this synthesis approach and they are given in Figures 7.9 and 7.10. The lowpass prototype cutoff frequencies are

$$\hat{\Omega}_p = \Omega_{p2} - \Omega_{p1} = 0.0555556$$

and

$$\hat{\Omega}_s = 0.1097430$$

The first example (see Fig. 7.9) was synthesized from a fourth-order cascade structure with direct form II sections. The second example (see Fig. 7.10) was synthesized from a fourth-order cascade structure with second-order sections in the form of Circuit E proposed by Avenhaus [34]. It was



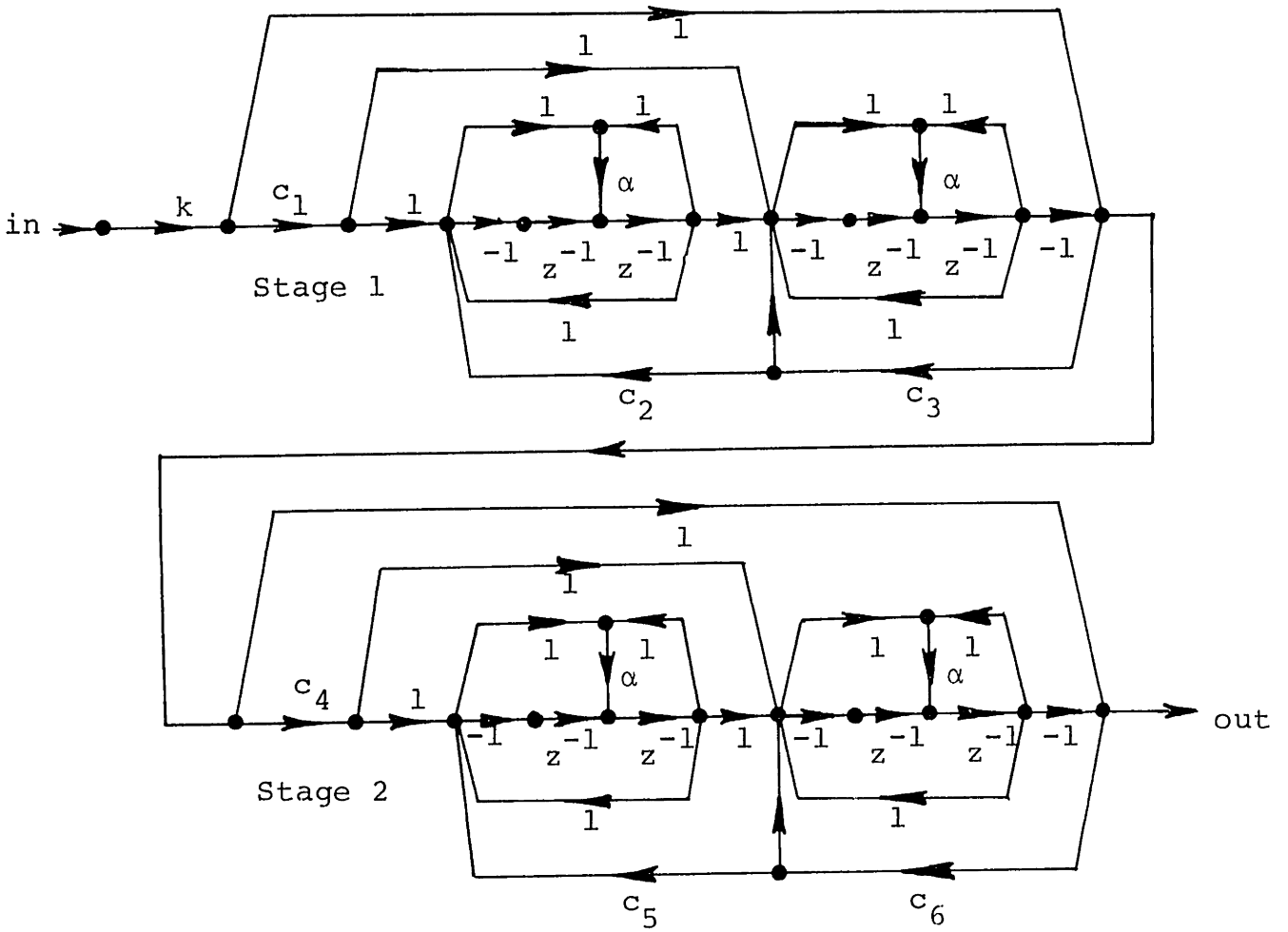
$k = 0.005656462366$ $\quad \alpha = 0.1915378547$

$c_1 = 1.907532841$ $\quad c_4 = 1.818129999$

$c_2 = -0.9410209204$ $\quad c_5 = -0.8324402465$

$c_3 = -1.865468242$ $\quad c_6 = -1.380195641$

Fig. 7.9 Cascade (direct form II) structure synthesized with the bandpass mapping transformation (Example 7).



$k = 0.005656462366$ $\alpha = 0.1915378547$

$c_1 = -0.134531758$ $c_4 = -0.619804359$

$c_2 = 0.3621618696$ $c_5 = 0.07868393672$

$c_3 = 0.0924671586$ $c_6 = 0.1818700012$

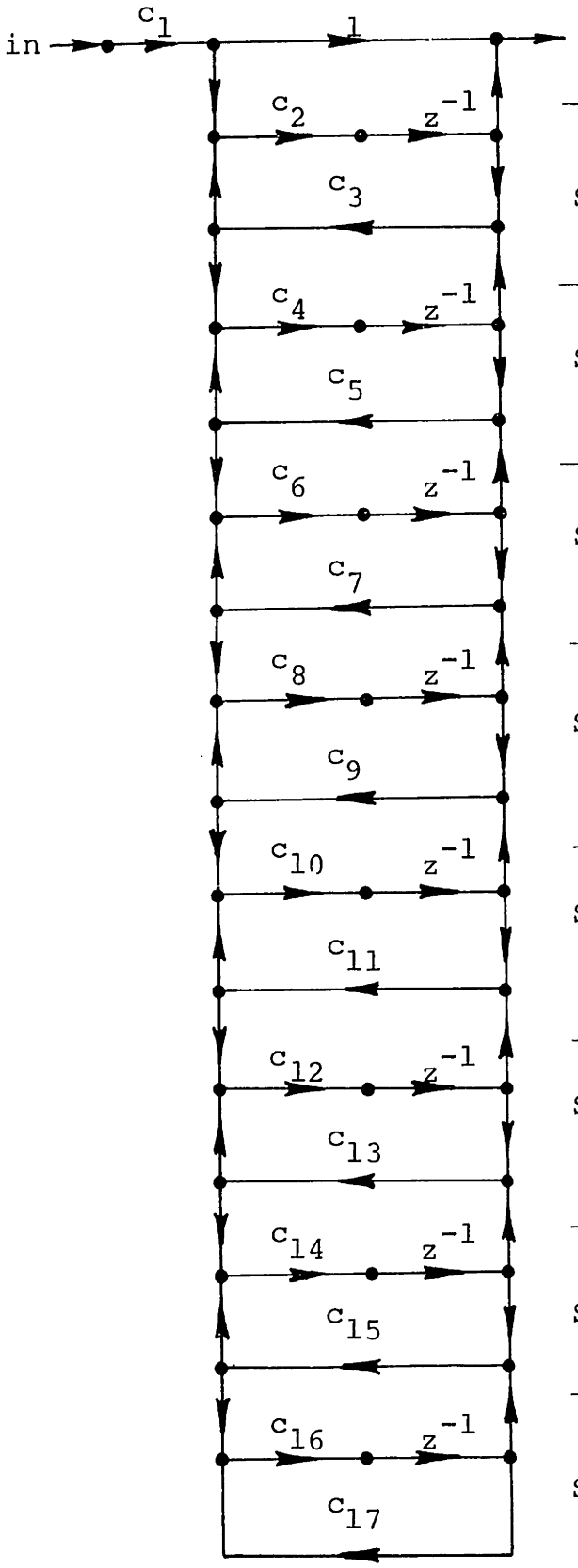
Fig. 7.10 Cascade (Avenhaus circuit E) structure synthesized with the bandpass mapping transformation (Example 8).

demonstrated by Avenhaus that the density of allowed pole positions for this circuit is relatively large near z = 1 and, therefore, this structure should have low coefficient sensitivity properties for narrow-band, lowpass filter designs. In both examples only 11 multiplies were required.

7.2.4 Continued Fraction Expansion Structures

Another way of representing the system function of a filter is in terms of continued fraction expansions. Mitra and Sherwood [31] have demonstrated that four basic forms of these expansions are possible and that from these continued fraction expansions we can construct a class of structures. Two of these types of structures, type IA and type IB, were synthesized for the eighth-order bandpass example and they are given in Figures 7.11 and 7.12, respectively. The continued fraction expansion of type IIA was not realizable for this example and the structure corresponding to type IIB is non-computable.

As we can see from the examples, these structures take the form of long "chains" and the attenuation in the stop band is basically obtained by cancellation of the two branch signals entering the output node at the top of the "chain". In effect, there is little isolation between the input and the output of the filter. Consequently, we might expect that the accuracy requirements on the coefficients of the structure are quite severe, especially in situations where we have



$c_1 = 0.005656462366$

$c_2 = 0.0919382834$
$c_3 = -49.83636203$

$c_4 = -0.1152951246$
$c_5 = 4.789010497$

$c_6 = -0.07982020996$
$c_7 = 16.77636663$

$c_8 = 0.1244561442$
$c_9 = -1.893836272$

$c_{10} = -0.00626801611$
$c_{11} = 3.833751971$

$c_{12} = 23.59416286$
$c_{13} = -3.825612822$

$c_{14} = 0.002102572645$
$c_{15} = -104.9491082$

$c_{16} = -0.02701849245$
$c_{17} = 75.86358186$

Fig. 7.11 Continued fraction expansion structure type 1A, (Example 9).

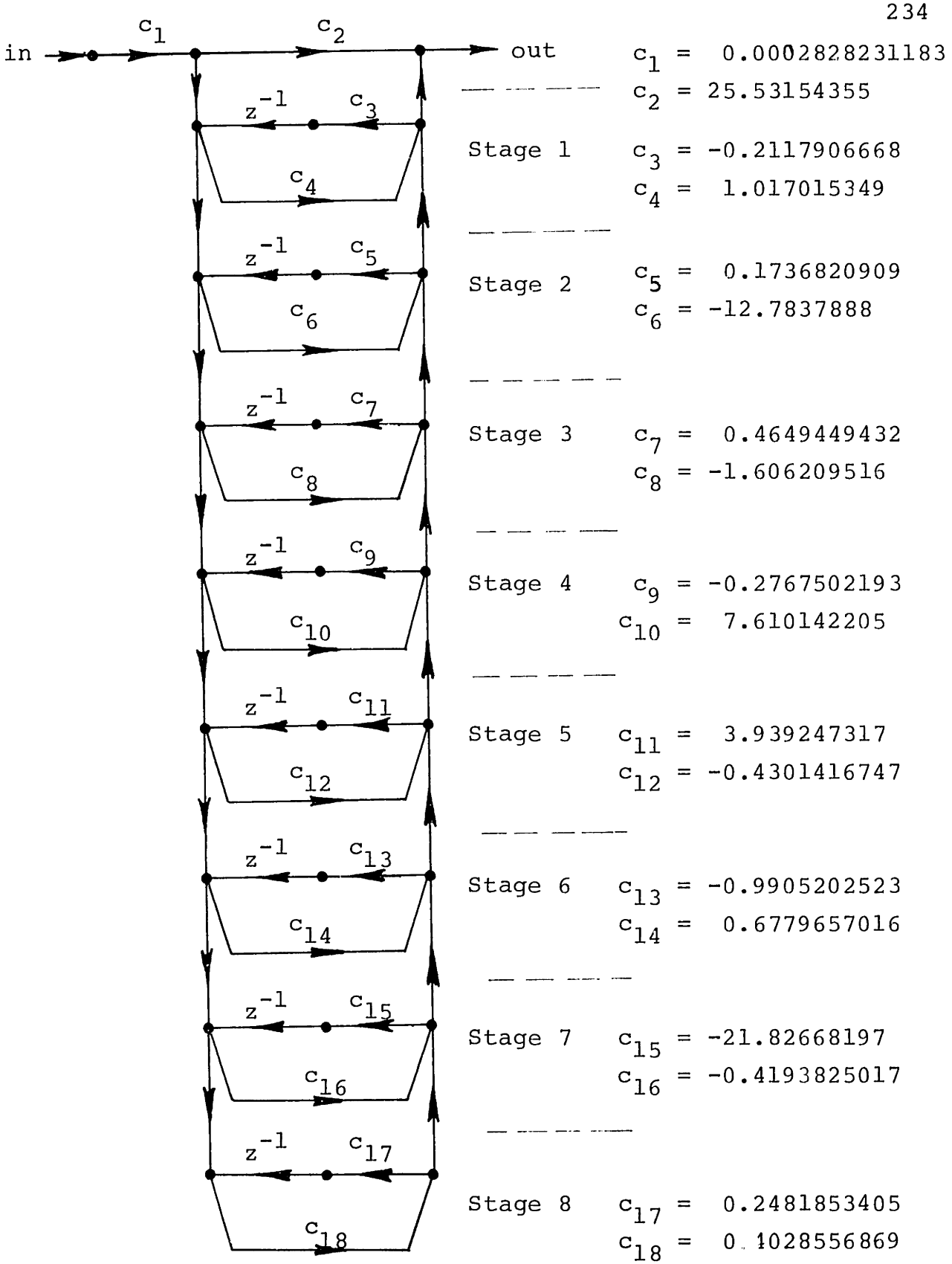


Fig. 7.12 Continued fraction expansion structure type 1B (Example 10).

high order filter functions which require high attenuations in the stop band.

7.2.5 Ladder and Lattice Structures

Another major category of structures which have received considerable attention recently are the ladder or lattice structures. Several approaches for the synthesis of these types of structures have been proposed by various authors [15], [16], [17], [18], [35], [36], [37], [64]. Unlike the previous categories of structures, these structures are not synthesized by representing the system function in terms of some form of factorization or expansion. Instead, the mechanisms behind the synthesis procedures for these structures are closely related to the two-port network synthesis techniques associated with conventional analog RLC circuits. Generally, the digital ladder and lattice structures are synthesized in terms of building blocks which take the form of two-input, two-output subnetworks where the inputs and outputs are grouped in terms of input-output "two-pairs". The building blocks are generally connected in a cascade configuration in terms of these "two-pair" connections (see Chapter 2, Section 2.8). Usually one of the signals of a pair can be associated with a forward directed signal and the other can be associated with a return signal or a feedback path. Consequently, the structures are composed of multiple, interconnected sets of feedback paths.

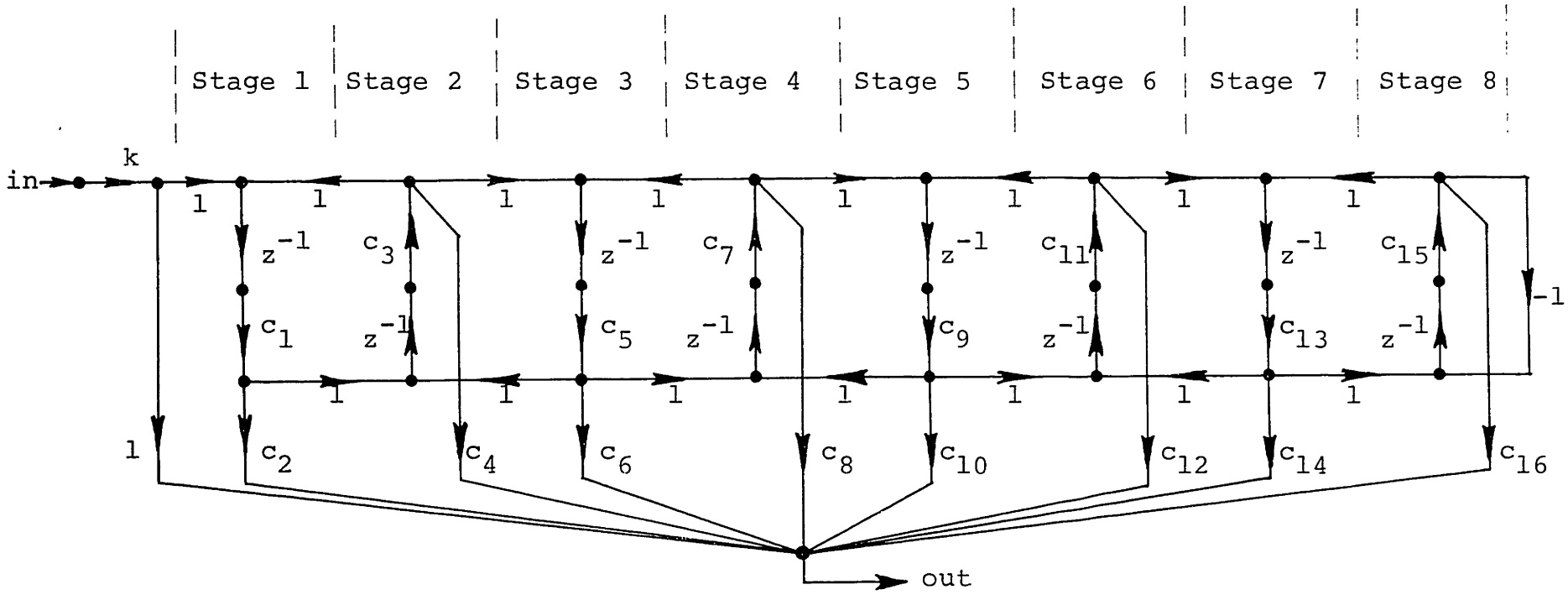
The ladder structure as proposed by Mitra and Sherwood [18] was synthesized and is given in Fig. 7.13. This structure has a ladder configuration only in terms of the implementation of the poles of the system function. The zeros are obtained as a linear combination of the internal variables or state variables in the ladder and are not generated by the internal process of the ladder itself. The part of the network which constitutes the ladder is synthesized by means of a continued fraction expansion in a manner not unlike that used in the continued fraction expansion structures in Figures 7.11 and 7.12.

A second type of ladder structure, synthesized according to the methods proposed by Gray and Markel [35], is given in Fig. 7.14. This structure is also a ladder structure only in terms of the implementation of the poles of the system function and the zeros are again obtained as a linear combination of the internal "state variables" in the structure. Two important distinctions can be made for this class of structures as observed by Gray and Markel. The coefficients associated with the "ladder part" of the structure are always less than one in magnitude for stable filter designs. Thus, there is a built-in upper limit on the range of these coefficient values. Secondly, there is a parity constant of $\pm$ 1 associated with each stage of the ladder. These parity constants can be chosen in the structure to maintain a proper scaling of the signal levels such that the dynamic range of the signals in the

structure is kept at a minimum. An optimization procedure for choosing these constants is given by Gray and Markel. Another important application of this structure and related structures, in addition to filter design, is the modeling of acoustic resonant systems such as the vocal tract in speech research [65].

Another procedure for the synthesis of ladder structures has been proposed by Fettweis [15], [16], [66], and a modification for implementing the zeros in these structures has been proposed by the author [36]. For this class of structures the synthesis procedure is not only similar to that for analog two-port circuits but, in fact, it is obtained by starting from an LC analog ladder design and appropriately transforming this design into a digital ladder design. Thus, all of the numerous tables and procedures which have been developed for the design of the LC ladder filters can also be used for the design of digital ladder filters as well. The resulting digital filter is in effect a digital simulation of the wave-flow diagram of the corresponding analog circuit and the frequency response of the analog circuit is mapped, according to the bilinear transformation, to the digital case. These types of digital filters, therefore, have characteristics which are similar to those of their analog counterparts.

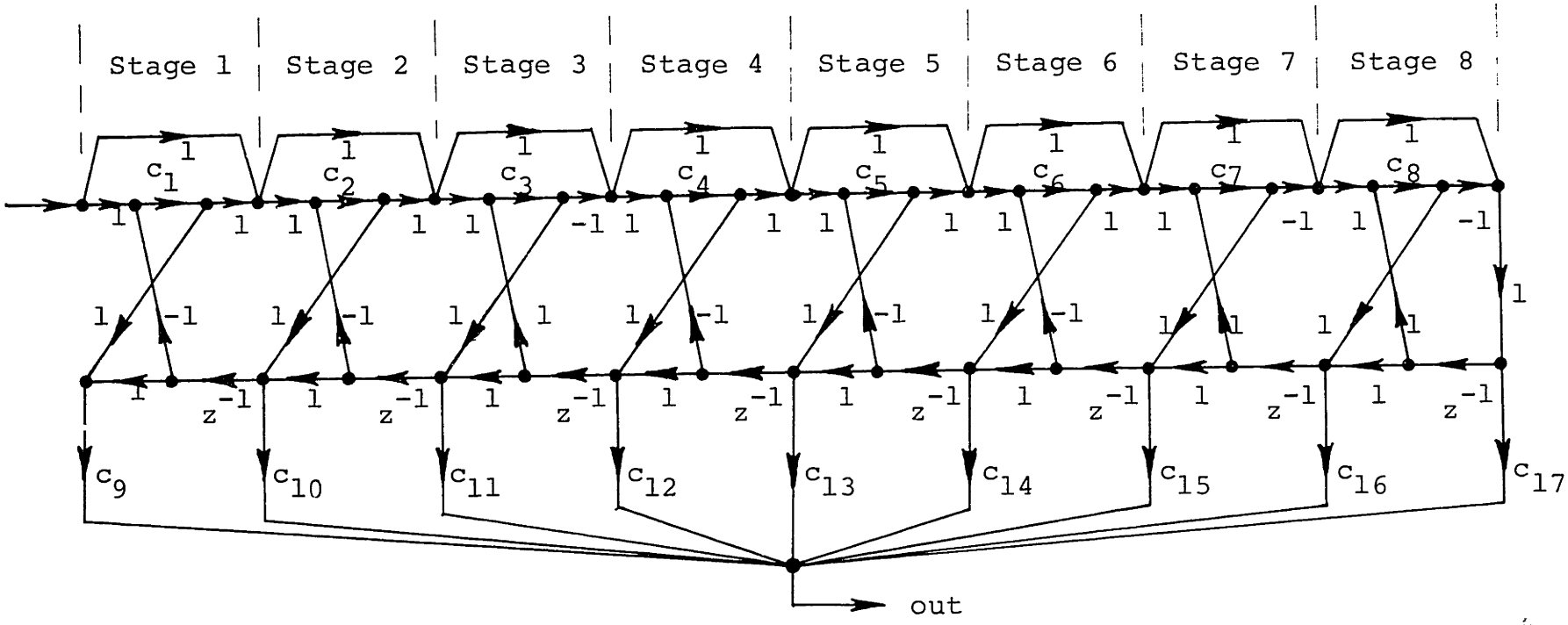
Two important properties of the doubly (resistively) terminated LC ladder structures are that they are passive and lossless [59]. Consequently, the power gain of these



$k = 0.005656462366$

$c_1 = 435.7362441$
$c_2 = 0.000210995263$
$c_3 = -0.00201410002$
$c_4 = 0.4799924366$
$c_5 = 12.16053691$
$c_6 = 0.03838502472$
$c_7 = -0.06547062948$
$c_8 = 0.5519655223$
$c_9 = 1.762511418$
$c_{10} = 0.3488742835$
$c_{11} = -0.3958370858$
$c_{12} = -0.8478865247$
$c_{13} = 1.085950607$
$c_{14} = -0.1967175203$
$c_{15} = -1.479757145$
$c_{16} = -0.1358646584$

Fig. 7.13 Ladder structure by Mitra and Sherwood (Example 11).

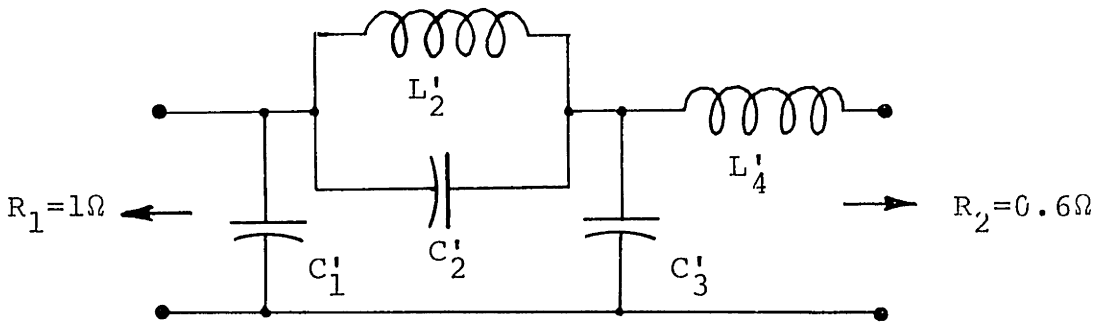


$c_1 = 0.7833447265$
$c_2 = -0.1885428229$
$c_3 = 0.9843741951$
$c_4 = -0.1920962931$
$c_5 = 0.9936376827$
$c_6 = -0.1890252997$
$c_7 = 0.9914182038$
$c_8 = -0.1964790194$
$c_9 = 0.005656462366$
$c_{10} = 0.0002916124024$
$c_{11} = -0.001699657657$
$c_{12} = -0.06980454614$
$c_{13} = 0.2031623313$
$c_{14} = 0.01061747099$
$c_{15} = -0.01641846145$
$c_{16} = -1.031187912$
$c_{17} = 0.8446328267$

Fig. 7.14 Ladder structure by Gray and Markel (Example 12).

structures can never exceed unity. From this property it can be shown [58] that for equiripple filters at the passband frequencies, where the gain actually achieves unity (the peak ripple frequencies), the sensitivity of this gain, with respect to variations of the filter elements, is zero. As a result, these types of analog structures are noted for their low sensitivity properties. It can be shown that a similar property also applies to the case of digital ladder structures which are patterned after these types of LC ladder structures [17], [58], [64]. Consequently, we can expect to achieve low coefficient word lengths with these types of digital filter structures.

To perform the synthesis of the eighth-order bandpass example we started with a fourth-order LC lowpass prototype (see Fig. 7.15(a)). This design was obtained using a standard set of tables by Saal [67] (pp. 32, circuit CO425b, $\Theta = 32$, $\Omega_s = 2.019399$, $A_s = 43.7$). This structure was then transformed to the bandpass structure in Fig. 7.15(b) using conventional transformations for LC ladder designs [67]. An important characteristic of this bandpass LC structure is that the gain of the structure is zero at DC and at infinity. This is in contradiction to the even order optimal bandpass elliptic designs which require that the gain at these frequencies be finite. In fact, it is not possible to realize the optimal elliptic designs for even order filters by using this class of LC structures (odd order optimal elliptic designs present no

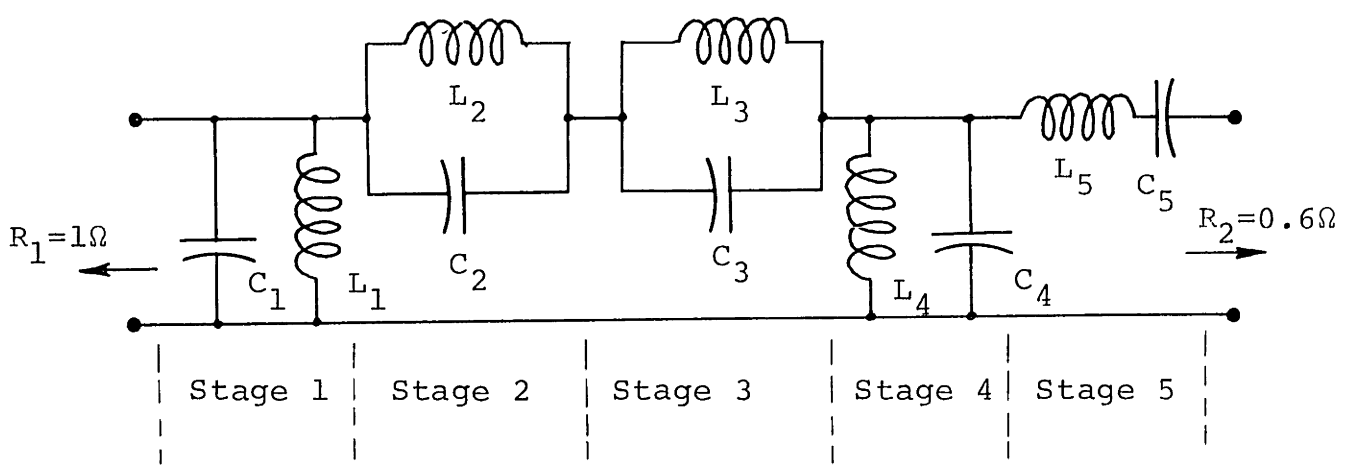


$C_1' = 1.258$ $\quad C_3' = 1.965$

$C_2' = 0.1937$ $\quad L_4' = 0.8605$

$L_2' = 1.073$

(a)



$C_1 = 7.05639$ $\quad C_3 = 2.68897$ $\quad C_5 = 0.207180$

$L_1 = 0.141716$ $\quad L_3 = 0.548493$ $\quad L_5 = 4.82673$

$C_2 = 1.82318$ $\quad C_4 = 11.02210$

$L_2 = 0.371889$ $\quad L_4 = 0.0907268$

(b)

Fig. 7.15 (a) Normalized fourth-order LC lowpass prototype
(b) Bandpass eight-order LC ladder structure.

problems). What is done in practice [59] is that the optimal even order elliptic designs are transformed to slightly less than optimal designs in a manner such that the largest zero in the frequency response is mapped to infinity and the smallest zero, for bandpass designs, is mapped to DC. This is illustrated in Fig. 7.16. Because of this problem we could not synthesize an eighth-order digital ladder structure of this type for the specifications given in Section 7.1. Instead, we used the nearest suboptimal eighth-order design in the tables by Saal [67] which corresponded to the LC structures in Fig. 7.15. The equivalent digital design specifications for this analog structure corresponded to:

$$\varepsilon_p = 0.01623 \ , \qquad \varepsilon_s = 0.0066705 \ ,$$
$$\Omega_{p1} = 0.4111111 \ \pi, \qquad \Omega_{s1} = 0.3839368 \ \pi,$$
$$\Omega_{p2} = 0.4666667 \ \pi, \qquad \Omega_{s2} = 0.4952625 \ \pi.$$

Using the techniques suggested by Fettweis and Sedlmeyer [15], [16] and the modification for implementing zeros suggested by the author [36], the analog ladder design of Fig. 7.15(b) was transformed to the digital ladder configuration in Fig. 7.17. Unlike the previous types of digital ladder structures in this section, this type of structure is a true ladder configuration in that both the zeros and poles of the filter are generated by the ladder mechanism. As in the case of its LC analog counterpart, the digital ladder structure in

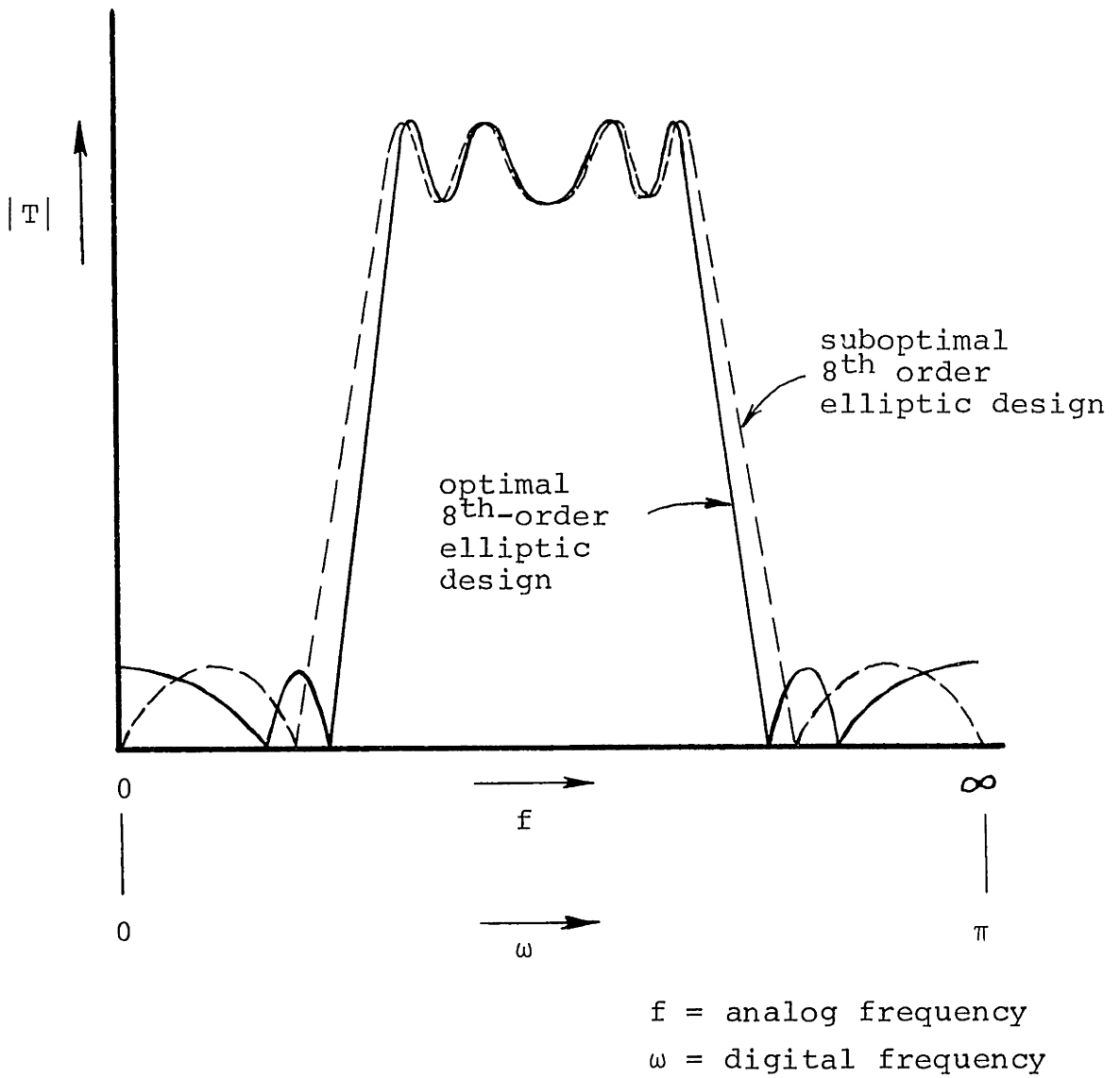
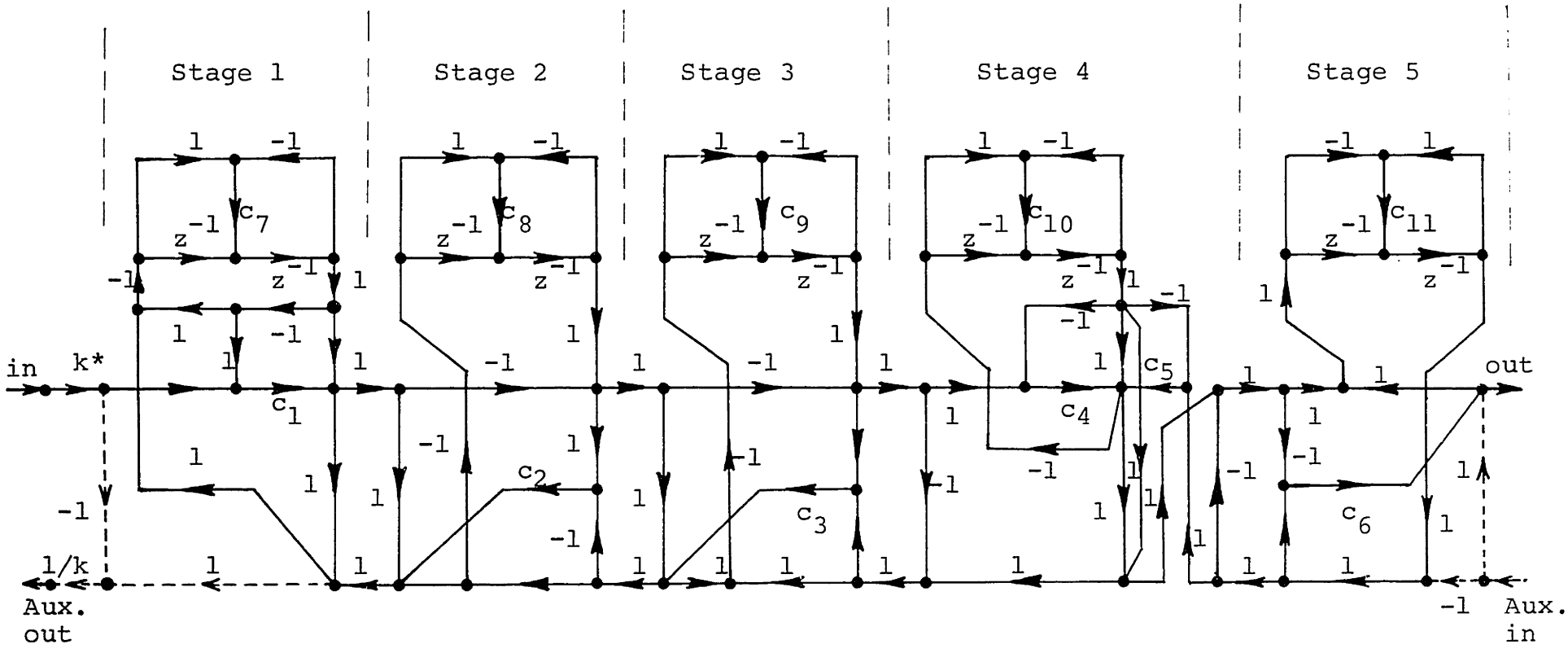


Fig. 7.16 Comparison of optimal eighth-order elliptic bandpass filter and suboptimal design necessary for LC ladder structure.



* see Text

$k = 1.31194961$	$c_5 = 0.007805575218$	$c_9 = -0.3698270573$
$c_1 = 0.06502370945$	$c_6 = 0.0574957168$	$c_{10} = -0.1915378541$
$c_2 = 0.2235692269$	$c_7 = -0.1915378541$	$c_{11} = 0.1915378541$
$c_3 = 0.5809284606$	$c_8 = 0.0003565321277$	
$c_4 = 0.1626985222$		

Fig. 7.17 Fettweis ladder structure (Example 13).

Fig. 7.17 can be used in either of two ways. The signal can be inserted at the left input of the structure in which case the passband frequencies will appear at the right output and the stop band frequencies will appear at the left auxiliary output. Alternatively, the signal to be filtered can be inserted in the auxiliary input on the right in which case the passband frequencies will appear at the auxiliary output on the left and the stop band frequencies will appear at the output on the right. In either case, the filter can be used simultaneously as a bandpass and a band stop filter. The gain constant K (see Fig. 7.16) can be evaluated as

$$K = (1 + \varepsilon_p) \sqrt{R_1/R_2} , \qquad (7.7)$$

where R_1 and R_2 are the input and output terminating resistances in the analog counterpart (see Fig. 7.15(b)). For most designs this value is close to 1, especially for odd order filter designs where $R_1 = R_2$ (even order suboptimal filter designs can also be synthesized such that $R_1 = R_2$ [59], [67]). Thus, for applications where the gain constant is not critical we may choose K to be 1. In this case, we can say that this class of structures is truly canonical (see Section 7.2.3) in terms of the number of multiplies.

7.3 COMPARISONS

A summary of the characteristics of the various structures

is given in Table I and an interpretation of these results is given below.

7.3.1 Coefficient Word Length

The comparisons of the coefficient word lengths of the various structures are made on the basis of the fixed-point statistical word length defined in Chapter 6. The reasons for using this word length definition are explained in Section 6.3. The statistical word lengths are given in all cases for relative errors R equal to 1 and confidence factors y equal to 0.95, i.e., x = 2 (see Chapter 6). Given these values, statistical word lengths for different values of R and y can always be determined using the definition of the statistical word length. The statistical word length for each of the structures was determined separately for the passband and stop bands in order to obtain a more general picture of the overall sensitivity properties of the structure. In the evaluation of the statistical word lengths of the various structures we did not include the coefficient sensitivity of the coefficient which corresponded to the gain constant as this coefficient does not actually contribute to the shaping of the frequency response.

For each structure the value i_M used in the statistical word length definition is given, where 2^{i_M} is the power of 2 represented by the most significant bit in the coefficient word length. This value i_M is, in general, different for

TABLE I

Example	Description of Network	Statistical Word Length (bits) Passband	Stopband	i_M	No. of Multiplies	No. of Adds	Bit x Multiply Product
1	Direct Form II	20.86	10.85	2	16	16	334
2	Parallel Form	10.12	7.95	-1	18	16	182
3	Cascade (Direct Form II)	11.33	5.76	0	13	16	147
4	Cascade (Coupled Form)	11.70	5.78	0	21	20	246
5	Cascade (Avenhaus B)	11.22	5.96	0	13	16	146
6	Cascade (Avenhaus F)	10.57	5.95	0	13	24	137
7	BP Transformed (Cascade, Direct Form)	12.69	7.23	0	11	16	139
8	BP Transformed (Cascade, Avenhaus E)	9.99	6.21	-1	11	20	110
9	Continued Fraction 1A	29.64	21.69	6	17	16	504
10	Continued Fraction 1B	22.61	14.46	4	18	16	408
11	Ladder (Mitra and Sherwood)	28.72	20.97	8	17	16	488
12	Ladder (Gray and Markel)	13.97	10.81	0	17	32	238
13	Ladder * (Fettweis)	11.35*	5.67	-1*	12/11*	31	136/125

* See text

different structures. It is chosen such that the largest value of the magnitudes of the coefficients in the structure is within the range 2^{i_M+1} and 2^{i_M}. One exception was made for the case of the Fettweis ladder structure of Fig. 7.17. In this structure the coefficient K exceeded this range. As discussed in Section 7.2.5, this coefficient is generally close to 1 and, for applications where the absolute gain of the filter is not critical, we can choose it to be 1. Alternatively, we can always avoid the problem of implementing the coefficient K by using the filter in reversed form. In this case, we use the auxiliary inputs and outputs and implement the coefficient 1/K for which there is no problem in choosing $i_M = -1$.

In the evaluation of the statistical word length we needed to know the maximum error frequencies ω_{pu}, $\omega_{p\ell}$, and ω_s for the bandpass elliptic example. These frequencies are:

Passband		Stop band
ω_{pu}	$\omega_{p\ell}$	ω_s
$0.413056\ \pi$	$0.411111\ \pi$	0
$0.427222\ \pi$	$0.418333\ \pi$	$0.366667\ \pi$
$0.450000\ \pi$	$0.438889\ \pi$	$0.384702\ \pi$
$0.464722\ \pi$	$0.458889\ \pi$	$0.494444\ \pi$
	$0.466667\ \pi$	$0.516667\ \pi$
		π

and for the special case of the slightly suboptimal elliptic design used for the eighth-order Fettweis ladder structure they are:

Passband		Stop band
ω_{pu}	$\omega_{p\ell}$	ω_s
$0.413056\ \pi$	$0.411111\ \pi$	$0.361111\ \pi$
$0.427778\ \pi$	$0.418333\ \pi$	$0.383937\ \pi$
$0.450000\ \pi$	$0.438889\ \pi$	$0.495262\ \pi$
$0.464722\ \pi$	$0.458889\ \pi$	$0.522222\ \pi$
	$0.466667\ \pi$	

For the case of the structures which were designed by the bandpass mapping circuits (see Figures 7.9 and 7.10), it was necessary to account for the fact that the coefficient α, although it appears in four different places in the structures, is the same coefficient. Thus, the sensitivity S in the statistical word length definition (see Chapter 6) is

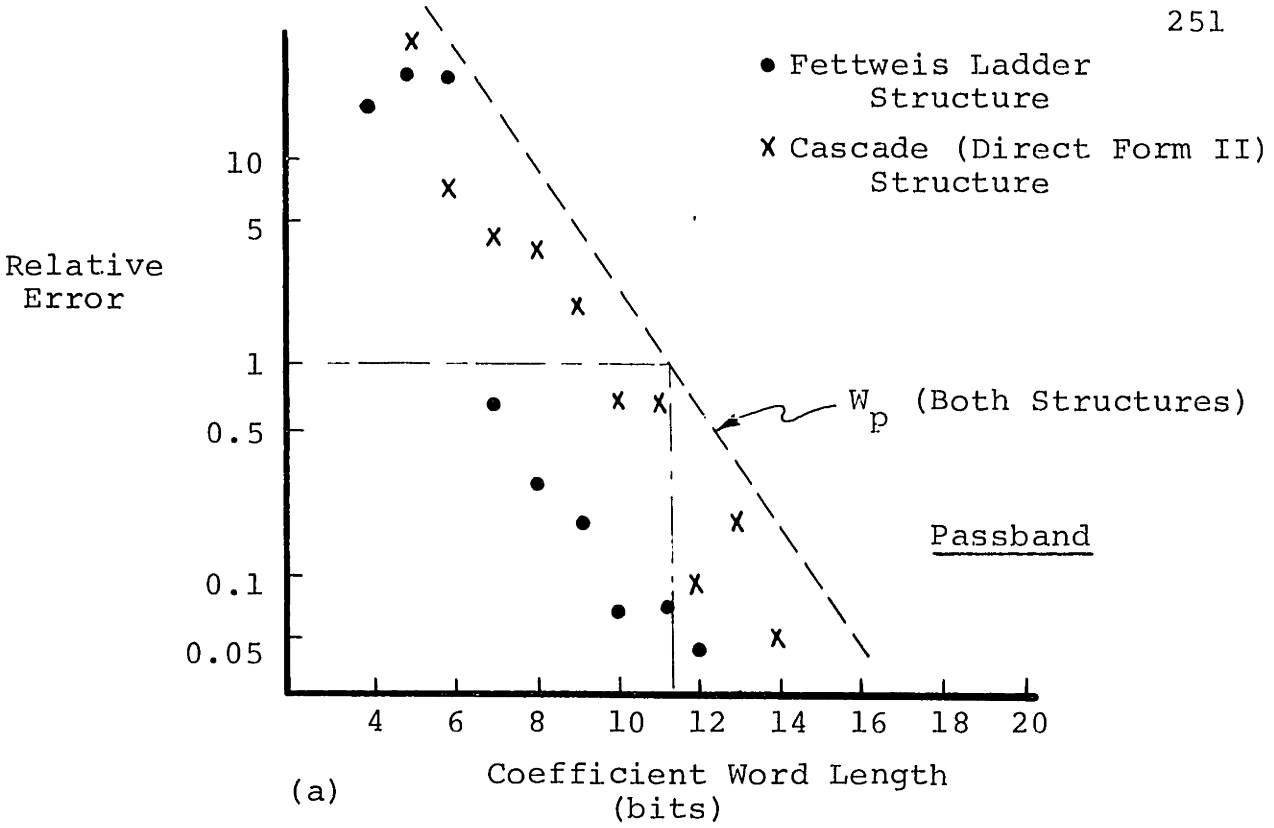
$$S = \left(\frac{\partial |T|}{\partial \alpha}\right)^2 + \sum_i \left(\frac{\partial |T|}{\partial c_i}\right)^2 .$$

$$\left(\begin{matrix}\text{Other} \\ \text{coefficients}\end{matrix}\right)$$

In general, we found for this bandpass example that the direct form structure, the continued fraction form structures, and the ladder structure of Mitra and Sherwood required considerably larger coefficient word lengths (see Table I). This was also observed in Chapter 6 (see Fig. 6.6) where the

coefficients of these structures were rounded and the actual errors in the system function were measured. The structures with the lowest coefficient word lengths were the parallel form, the cascade (Avenhaus F) form, and the bandpass transformed structure designed from a cascade (Avenhaus E) lowpass prototype.

The statistical word length for the Fettweis ladder structure is essentially the same as that of the cascade (direct form II) structure in both the passband and the stop band (see Table I). However, when we rounded the coefficients in fixed-point arithmetic, we found that in the passband the Fettweis ladder structure required about three bits less for a given relative error than the cascade structure, provided that the coefficients were not "grossly quantized" such that the relative errors exceeded approximately 1 or 2. This can be observed in Fig 7.18(a). For the stop band, as seen in Fig. 7.18(b), the word lengths are about the same. Thus, it appears that the statistical word length for the Fettweis ladder structure is somewhat overestimated in the passband. This may be attributed to the fact that in the statistical word length model we assumed that positive and negative errors in the system function due to coefficient errors were equally possible. In practice, it appears that due to the "pseudo-passive" and "pseudolossless" properties [58] of the Fettweis ladder structure, which are digital simulations of their passive and lossless LC analog counterparts, positive and



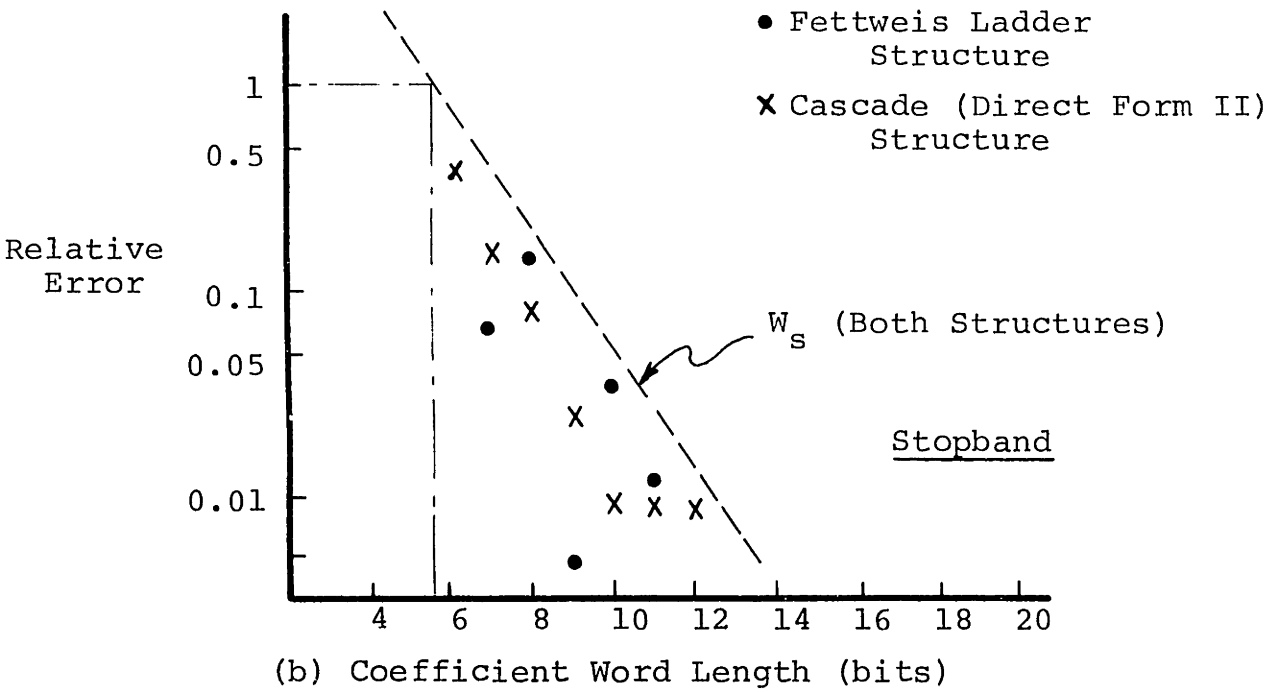


Fig. 7.18 Comparisons of the actual coefficient word length for the Fettweis ladder structure and the cascade (direct form II) structure, (a) passband, (b) stop band.

negative errors are not equally possible.

7.3.2 Multiplies, Adds, and Bit·Multiplier Products

The number of multiplies and adds required for each structure is tabulated in Table I. The bandpass transformed structures and the Fettweis ladder structure could be implemented with 11 multiplies each and are, therefore, truly canonical according to the definition in Section 7.2.3. For the cascade structures, except the cascade (coupled form) structure, the number of multiplies was 13. The remaining structures generally required about 17 multiplies which is the number of degrees of freedom necessary to specify the poles and zeros of an arbitrary eighth-order system function where complex poles and zeros are in conjugate pairs.

The number of adds in many of the structures was 16. For those structures where we could obtain lower coefficient word lengths than in the cascade (direct form II) structure (e.g. Examples 6, 8, and 13), we found that it was generally obtained at the expense of increasing the number of adds and the complexity of the structure.

To obtain a crude estimate of the total amount of computation involved in each structure we evaluated the bit·multiplier products (see Table I). The number of bits was taken as the passband statistical word length.

7.3.3 Modularity and Complexity of the Structures in Terms of Hardware

The modularity in a digital structure can be defined as the topological redundancy in the structure. Most structures exhibited a form of modularity which is identical from stage to stage. The only exception to this is the Fettweis ladder structure in which only some of the stages are identical. The bandpass transformed structures exhibited a dual form of modularity. They are identical from stage to stage and also in the bandpass mapping networks which replace the delays in the lowpass prototype.

In the implementation of a structure in hardware it is often desirable to incorporate the modularity or topological redundancy of the structure into the hardware design. In this way we can reduce the amount of hardware and greatly simplify the control of the hardware, particularly if the modularity is identical from stage to stage. In effect, the structure is implemented on a stage by stage basis in hardware. If, however, the modularity of the structure is in terms of two-input, two-output stages, as in the ladder structures, and there are direct paths from the inputs to the outputs, then the approach does not work. This occurs in the case of the Fettweis ladder structure in Fig. 7.17. In this structure the computations involved in any stage cannot be completely performed without first obtaining an input signal from another stage. Thus, we expect that a hardware

implementation of this structure would be considerably more complex than hardware implementations of, for example, the cascade structures.

7.3.4 Other Comments

The comparisons of the structures in this chapter have been made primarily on the basis of coefficient word length and on the number of multiplies and adds required for the structures. These attributes are clearly important in the choice of a filter structure. However, we do not want to leave the reader with the impression that these are the only factors to be considered. Of equal importance are the issues of roundoff noise, dynamic range, and limit cycle effects. Considerable work has been done in these areas by Oppenheim, Weinstein, Kaiser, Jackson, Liu, and others [3], [4]. Although these issues have not been considered in this work, we feel that it is important to mention them in order that the results of this work are viewed in proper perspective to the overall issues of filter structure synthesis.

implementation of this structure would be considerably more complex than hardware implementations of, for example, the cascade structures.

7.3.4 Other Comments

The comparisons of the structures in this chapter have been made primarily on the basis of coefficient word length and on the number of multiplies and adds required for the structures. These attributes are clearly important in the choice of a filter structure. However, we do not want to leave the reader with the impression that these are the only factors to be considered. Of equal importance are the issues of roundoff noise, dynamic range, and limit cycle effects. Considerable work has been done in these areas by Oppenheim, Weinstein, Kaiser, Jackson, Liu, and others [3], [4]. Although these issues have not been considered in this work, we feel that it is important to mention them in order that the results of this work are viewed in proper perspective to the overall issues of filter structure synthesis.

Chapter VIII

SUMMARY AND SUGGESTIONS FOR FURTHER RESEARCH

8.1 SUMMARY

In this dissertation we have investigated the problems of analysis and synthesis of digital filter structures. Primary emphasis has been placed on the issues of coefficient word length. We began by developing a rigorous network theory formalism for digital filters. A central part of this formalism is a general matrix representation of digital networks which has the property of being unique for a given structure. Using this network theory formalism, we then derived a number of general theorems and properties of digital networks, some of which are new and some of which were originally proposed by others.

In the second phase of the thesis we showed how the matrix representation and the network properties could be effectively used in a computer-aided network analysis context. As a part of the thesis, a general purpose computer-aided digital network analysis package, CADNAP, was developed using these concepts. CADNAP was used extensively in the analysis of examples throughout the thesis.

In the third phase of the thesis we expanded upon the concepts of a statistical estimate of the coefficient word length of a digital filter. We then developed an optimization

procedure for minimizing this statistical word length measure and showed that it generally leads to a reduction in the actual coefficient word length of the filter by about one to three bits. The procedure is based on matching the approximation of the filter to the specific coefficient sensitivity properties of the structure by which it is implemented.

In the final phase of the thesis, we investigated the problem of choosing a filter structure. Thirteen different types of filter structures were analyzed and compared on the basis of coefficient word length, the number of required multiplies and adds, and other hardware-related issues. These structures were chosen to represent a typical cross section of the various known methods and philosophies for filter structure synthesis.

8.2 SUGGESTIONS FOR FURTHER RESEARCH

There are a number of directions and issues which are of interest for further research. In the area of computer-aided analysis of digital networks we briefly discussed, in Section 5.3, the use of sparse matrix methods in solving the set of linear simultaneous equations for frequency response analysis. Alternatively, another approach was to first calculate the poles and zeros of the system function of the structure and then compute the frequency response from these. It would be of interest to try these approaches in the computer-aided analysis package. Also, it would be of interest to generate

other types of analysis features for CADNAP such as the analysis of limit cycles, roundoff noise, dynamic range, etc.

Another area for further research is in the application of the statistical word length optimization procedure. Several conjectures are made in the conclusions of Chapter 6, Section 6.6. These conjectures need to be explored in further detail. One is that this optimization procedure might be useful as an initial phase of a two-phase, nonlinear optimization procedure for finding the actual global minimum coefficient word length. A second conjecture is that for algorithmic filter synthesis, the statistical word length optimization procedure might be incorporated directly into the algorithmic design. In such algorithmic procedures we are often allowed to incorporate weighting factors at various frequencies into the design. These weighting factors could be selected in such a way that the statistical word length is minimized. A third area for further research is in the design of analog filters with a maximum allowed tolerance in the components. The concepts used for the statistical word length minimization procedure should be easily adaptable to the case of analog filters as well.

Another direction of continuing research is in the area of filter structures. It would be of interest to find other classes of structures which would be good candidates for filter design. Also, further investigation needs to be carried out in comparing the various structures in terms of

other properties such as roundoff noise, dynamic range, and limit cycles.

Appendix A

A USER'S GUIDE FOR THE COMPUTER-AIDED DIGITAL NETWORK ANALYSIS PACKAGE (CADNAP)

CONTENTS

A.1 INTRODUCTION

CADNAP is a set of general purpose programs for manipulating and analyzing digital networks. It is capable of handling digital structures of any arbitrary configuration and is designed to be highly interactive with the user. In its present state of development, CADNAP contains operators to analyze network response functions from any point in a network to any other point. It can compute unit sample responses, frequency responses, and coefficient sensitivities of any transfer function in a network. Coefficients can be rounded or truncated in fixed-point arithmetic and structural manipulations such as taking the transpose of a network can be performed.

CADNAP is composed, for the purpose of flexibility, of a set of independent programs or modules. In this way individual modules can be used independently of the rest of the package to conserve computer storage. They can also be used as separate programs or subroutines for performing specialized analysis operations in other programs. Another advantage of the modular approach is that it makes CADNAP easily adaptable to modification and addition of new features.

The basic functional approach used by CADNAP is depicted in Fig. A.1. The common parameter in all of the operators is a network. A network is stored in compact form on the computer as a vector. The structure is reconstructed, in terms of its matrix representation, from this vector only when it is needed.

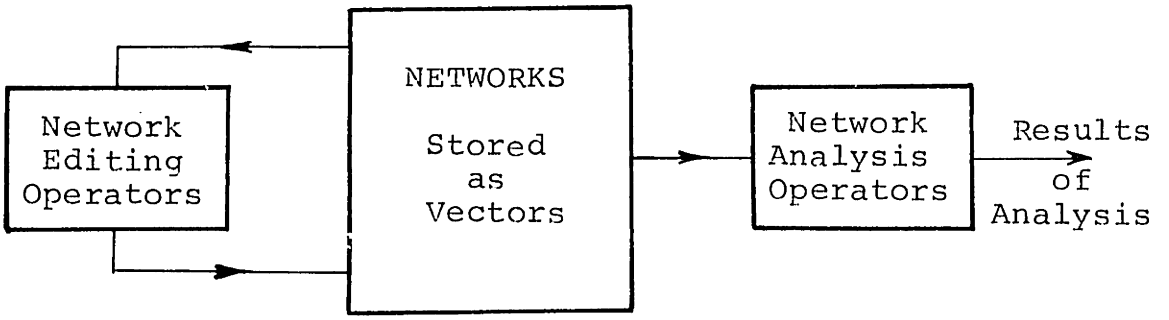


Fig. A.1 Basic functional chart for CADNAP.

Fig. A.2 APL keyboard.

In this way several networks can be stored simultaneously without consuming excessive amounts of storage. The operators or modules can be classified into two types: editing operators and analysis operators. The editing operators take a network, modify it in some way, and return a new network. This new network can replace the old one or can be redefined as a new network, in which case the old unmodified network is kept unchanged. The analysis operators take a network, perform some sort of analysis on it, and return the results of that analysis. The network remains unchanged and can be used again for other analyses or modifications.

A.2 THE BASE LANGUAGE OF CADNAP

The language that CADNAP is written in is the APL language [54], [55]. This language was chosen because it permits a high degree of user/computer interaction on a time-shared basis, which is necessary for a general analysis package of this type. In particular, the array features and the operator notation of APL were found to be very desirable in the development and the use of CADNAP. A second reason for choosing the APL language was because other programs for the synthesis of digital filters are available in APL [56] and the CADNAP program serves to compliment these programs by providing a more complete overall synthesis and analysis tool for digital signal processing problems.

CADNAP is designed as a logical extension of the APL

language. Consequently, a few basic concepts of the APL language are incorporated in the functional style of CADNAP and must be understood in order to use it. These concepts are outlined briefly in section A.3. A detailed knowledge of the APL language, while helpful, is not necessary unless the user is interested in further computations or analyses beyond the scope of CADNAP.

A.3 SOME FUNDAMENTALS OF THE APL LANGUAGE

A.3.1 The Keyboard

The APL language uses a special typeball and special keyboard symbols on the time-shared terminal. These symbols and their locations on the keyboard are given in Fig. A.2.

Corrections can be made, when typing a line, by back-spacing to the point of the error and pressing the ATTN key. The computer will respond by typing the symbol, ∨, under the error and spacing the paper up an additional line. You can then resume typing the line, with corrections, from that point. This procedure can be repeated as often as necessary until the line is typed in correctly.

A.3.2 Number Representations and Arrays in APL

Numbers in APL can be represented as integers, decimals, or exponents. For example, the number 253 can be expressed in either of the forms:

253 , 253.00 , 2.53*E*2 .

Negative numbers are preceeded by a raised minus sign (located above 2 on the keyboard). This sign should not be confused with the ordinary minus sign used as the subtraction sign. Some examples are:

```
¯253    ,    ¯0.253    ,    ¯2.53E¯2    .
```

APL is also capable of handling vectors, matrices, and higher order arrays of numbers. For example, a vector of 4 elements might be:

```
3   2.5   ¯4.1   2.1
```

Note that each number in the vector is separated by a single space. An example of a 4 row 3 column matrix might be given as:

```
 ¯1      2.3     3.4
 ¯1.5    2       3
 ¯8      0       ¯4.2
 ¯5      2       ¯3
```

An example of an array of 3 planes, 2 rows, and 2 columns might look like:

```
 2       ¯4
 3        5

 ¯2.3     3
  1       2

  1       5
  0       1.3
```

APL is also capable of recognizing vectors which have no elements in them, that is "empty vectors". An empty vector can be specified by the symbol ⍳0. The use of this special type of vector will become apparent later.

A.3.3 Variables

Variables are names of numbers or arrays of numbers. These names may be of any length and may contain letters or digits so long as the first character of the name is a letter. Variables are defined or changed by assigning a value to them with the aid of the back arrow symbol ← . For example, a scalar variable *A* can be defined to have a value of 5 by typing the following instruction

```
A←5
```

Variables can also represent arrays of numbers as well as single numbers. For example, a vector variable *B* can be defined as:

```
B←1 3 2 ¯3.8
```

The numerical value of a variable can be obtained by typing the variable name. The computer will respond by typing its value. For example

```
A                                    (user
5                                    (computer
B                                    (user
1 3 2 ¯3.8                           (computer.
```

If only specific element values of a vector are desired, these values can be obtained by specifying the indices of those elements in brackets. For example, *B*[2] gives the value of the second element in vector *B*. That is,

```
B[2]                                    (user
3                                       (computer.
```

More than one element can be specified by giving a vector of indices.

```
C←B[4 3]                                (user
C                                       (user
 3.8 2                                  (computer
```

This extends to higher order arrays as well. If *D* is the array

```
1 ¯3  5
2 ¯1  4
```

then *D*[2;3] is

```
4
```

and *D*[2;1 3] is

```
2  4 .
```

A vector can also be assigned to be an "empty vector" by making an assignment such as

```
A←⍳0
```

Another type of variable which will be used in CADNAP is the character vector. A character vector is a vector of literal characters (they do not have to be separated by spaces). To define a set of literal characters as a character vector it is necessary to precede and follow the characters by quotes. Both quotes must be used. For example, to assign the 5 element character string, *ABC*35 to a variable with the name *CHAR* we can type the instruction

```
CHAR←'ABC35'
```

If we request the value of *CHAR*, the computer will return the character string (without the quotes). For example,

```
CHAR                                    (user
ABC35                                   (computer.
```

A.3.4 Operator Notation in APL

An APL operator or function* can be one of two types, a monadic or a dyadic operator. If it is monadic, it operates on whatever variable is to the right of it (scalar, vector or array). It may or may not return another variable (scalar, vector or array) depending on how it is defined. If it does not return a variable then it will perform its function and stop. If it returns a variable, this variable can be assigned a variable name with the aid of a back arrow. For example,

---

*The terms "operator" and "function" will be used interchangeably.

let *A* be a variable to be operated on and *OPR*1 be a monadic operator with no return, then the use of *OPR*1 to operate on *A* would look like this:

```
OPR1 A
```

Now let *OPR*2 be a monadic operator with a return and let *B* be the name of the variable that we want to call this return. The use of *OPR*2 will then look like this:

```
B←OPR2 A
```

If we want to operate on *B* with *OPR*1 we can do the following:

```
B←OPR2 A
OPR1 B
```

Or we can achieve the same result this way:

```
OPR1 OPR2 A
```

This can be done because APL always starts at the right and works to the left. First *OPR*2 operates on *A* and returns a result. Then *OPR*1 takes this result and operates on it. In other words, operators which return results can be cascaded. For example, let *OPR*3 and *OPR*4 be monadic operators which return results, then any of the following sets of operations can be performed.

```
OPR1 OPR2 OPR3 OPR4 B
C←OPR2 OPR3 B
A←OPR2(OPR3 OPR4 B)
```

But this will not work,

C←*OPR*2 *OPR*1 Does not work!

because *OPR*1 does not return a result and *OPR*2 does not know what to operate on.

What if we attempt to use *OPR*2 in this manner?

*OPR*2 *A*

The computer will not know where to assign the result of *OPR*2 so it will type it out and not assign it to anything.

Dyadic operators work in much the same way as monadic operators except that they require two variables (scalars, vectors or arrays) to operate on, one to the right and one immediately to the left. Let *OPR*5 be a dyadic operator which does not return a result. Then the use of *OPR*5 to operate on two variables will look like this.

*A OPR*5 *B*

Now let *OPR*6, *OPR*7 and *OPR*8 be dyadic operators which return results and *C*, *D*, *E*, *F* be variables (scalars, vectors or arrays). Then the following operations are valid.

C←*A OPR*6 *B*
C←*A OPR*7 *B OPR*8 *D*

This last operation is the same as:

C←*A OPR*7 (*B OPR*8 *D*)

because *OPR8* takes *B* for its left argument and *D* for its right argument and returns a result which is then the right argument for *OPR7*.

The following operation can also be performed,

E←(A OPR6 B)OPR7 C

which is equivalent to the two statements

F←A OPR6 B
E←F OPR7 C

Note that the parentheses determine the order of operation.

A.3.5 Primative Operators in APL

Certain types of operators in APL are fundamental to the APL language and serve as the basis of all APL programs. These operators are known as primative operators or primative functions and they are given special symbols on the APL keyboard. All other operators will be referred to as programed operators because they are generated by programing sets of primative operators to perform specialized operations. Programed operators do not have special symbols on the keyboard. Instead, they take names similar to that of the variables. All of the operators in CADNAP are programed operators.

Some examples of primative operators are +, -, ×, and ÷, which, when used as dyadic operators, represent addition, subtraction, multiplication, and division, respectively. For

example, two scalar variables *A* and *B* can be added and assigned to *C* by typing the instruction

```
C←A+B
```

If *A* and *B* are vectors or arrays, they must be of the same order and size. The result of the operation will yield an array *C* which is of the same order and size as *A* and *B* and the elements in *C* will be the sums of the corresponding elements of *A* and *B*. For example:

```
C←3 4 5+7 ¯2 2                    (user
C                                 (user
10 2 7                            (computer.
```

If *A* and *B* are arrays of different sizes the operators +, -, ×, and ÷ will be invalid because the corresponding arrays *A* and *B* are not compatable. One exception to this rule is when one of the arrays is a scalar or a vector with a single element. In this case, the scalar operates on each of the elements of the array. For example,

```
3×2 1 4 ¯3                        (user
6 3 12 ¯9                         (computer,
```

but this will not work

```
3 2÷1 4 6 2          Not compatable!
```

A WORD OF CAUTION is necessary at this point. As stated

in section A.3.4, operators can be cascaded, in which case APL performs the operations from right to left. Thus the sequence of operations

```
C←3×4+2
```

is equivalent to 3×(4+2), Not (3×4)+2. The user must be aware at all times that the sequence of events, unless changed by using parenthesis ( ), is from right to left regardless of whether it agrees or violates the ordinary rules of multiplication and division preceeding addition and subtraction.

A list of some of the primative scalar operators which may be useful in manipulating data in CADNAP is given below. Some of the operators are monadic and some are dyadic.

Operator	Result of Operation
$X+Y$	X plus Y
$X-Y$	X minus Y
$-Y$	minus Y
$X\times Y$	X times Y
$X\div Y$	X divided by Y
$X*Y$	X to the Y^{th} power
$*Y$	e to the Y^{th} power
$\lvert Y$	absolute value of Y
$X\circledast Y$	log of Y to the base X (the symbol ⊛ is formed by overstriking o with *)
$\circledast Y$	natural log of Y

$\circ Y$	Pi times Y	
$1\circ Y$	sine of Y	(angles must be in radians)
$2\circ Y$	cosine of Y	
$3\circ Y$	tangent of Y	
$^{-}1\circ Y$	arcsine of Y	
$^{-}2\circ Y$	arcos of Y	
$^{-}3\circ Y$	arctan of Y	

Other operators which are useful for manipulating arrays are:

Operator	Result of Operation
$X\rho Y$	reshape Y to have dimensions X
ρY	dimension (size) of Y
$X[Y]$	the elements of X at locations Y
ιY	generates a vector of integers 1 2 3...Y
X,Y	Y catenated to X
$,Y$	ravel of Y (make Y a vector)
$X\uparrow Y$	take the first X elements of Y (or the last X elements of Y if X is a negative integer)
$X\downarrow Y$	drop the first X elements of Y (or the last X elements of Y if X is a negative integer)
$X\leftarrow Y$	assigns to X the value of Y

Examples:

Let A be the array

```
1   3   1
4   1.5 ¯1
```

and B be the array

```
1 2 4 6 8
```

Then ⍴A is: 2 3

⍴B is: 5

5⍴A is: 1 3 1 4 1.5

1 2⍴B is: 1 2

⍳5 is: 1 2 3 4 5

1 3,B is: 1 3 1 2 4 6 8

,A is: 1 3 1 4 1.5 ¯1

3↑B is: 1 2 4

¯3↑B is: 4 6 8

2↓B is: 4 6 8

¯2↓B is: 1 2 4

For a more detailed description of the above operators and a more extensive list of operators the reader is referred to any of the users guides to APL [54], [55].

A.3.6 System Operation and System Commands in APL

The basic structure of an APL system is depicted in Fig. A.3. The active workspace is the workspace and storage space

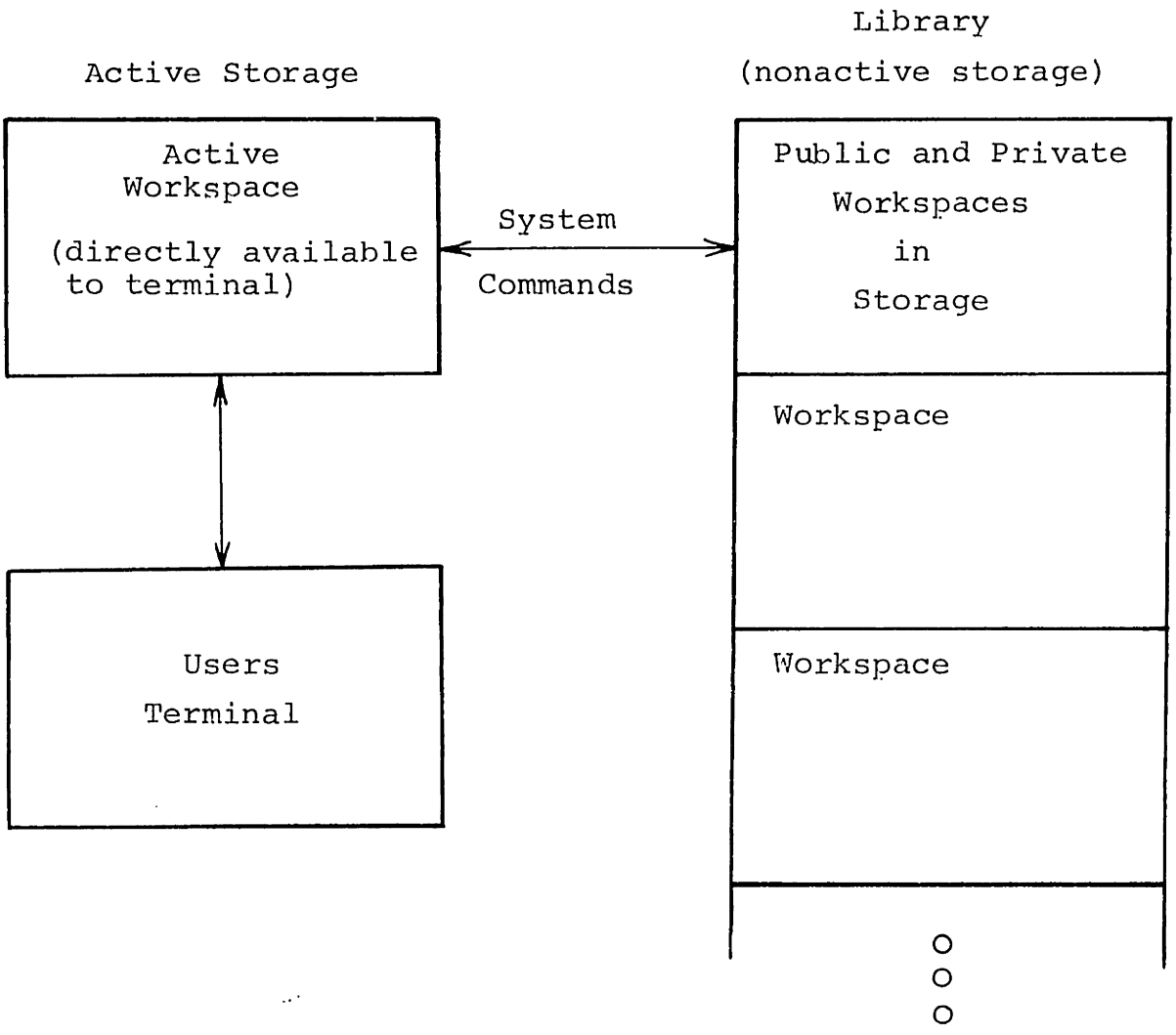


Fig. A.3 Structure of an APL system.

immediately accessable to the users terminal. Any operators or variables which the user is actively using must be located in the active workspace. Also, all computations and intermediate computations performed are done in the active workspace. Consequently, the amount of storage available in the active workspace governs how much storage the user has at his disposal. Typically, this value might be about 32 kilobytes for small workspaces.

The library provides additional nonactive storage for the user. All storage is arranged according to workspaces and associated with each workspace is an assigned name. For public workspaces which are accessable to many users, a number and a name are required to identify the workspace.

The system commands enable the user to obtain programs or workspaces from the library or store new workspaces in the library. They also allow the user to get listings of variables or programs in the active workspace or the library. All system commands are preceeded by a single right parenthesis. The following commands will be helpful for the CADNAP user:

)CLEAR

- erases everything from the active workspace except primative operators.

)LIB

- gives a listing of the users private libraries.

)OFF

- allows the user to log out of the system.

)FNS

- gives a listing of all programed operators (functions) in the active workspace.

)VARS

- gives a listing of all variables in the active workspace

)ERASE (names of variables, operators, or groups)

- erases variables, programed operators, or groups from the active workspace. For example, to erase a variable called *ALPHA* from the active workspace you would type the command

)ERASE ALPHA

)WSID

- gives the name of the active workspace.

)WSID (name)

- changes the name of the active workspace to the new name given. For example, to give the active workspace the name *WKSP*1, type

*)WSID WKSP*1

)*LOAD* (name of library)

- clears the active workspace and then loads a copy of the workspace from the library whose name is given in the command. For example, to load into the active workspace a copy of the library workspace *WKSP1*, type

)*LOAD WKSP1*

If you want to load a workspace from the public library, its appropriate number and name must be used. For example,

)*LOAD* 103 *WKSP1* .

)*COPY* (name of library) (name of variable, operator or group)

- copies the variable, operator, or group from a library into the active workspace. For example, to copy a variable *ALPHA* from a public workspace might look like this:

)*COPY* 103 *WKSP1 ALPHA* .

)*SAVE* (name of workspace)

- makes a copy of the active workspace and stores it in the user's private library. The name of this new library will be that given in the command. For example, to save a copy of the active workspace in the library and give it the name *WKSP2*, you would type the command:

)*SAVE WKSP2*

The most recently saved workspace replaces any earlier workspace of that name in the library.

)*DROP* (name of workspace)

- drops the specified workspace from the library (private files only).

)*GRP* (name)

- gives a listing of all variables and operators listed under the group name. Some variables and/or operators in CADNAP are listed in groups to reduce the number of commands necessary to *COPY* or *ERASE* them.

)*GRPS*

- gives a listing of all groups in the active workspace.

A.3.7 Errors and Further Reference

If an error is encountered in APL, the system will respond with an appropriate error message. Until you have learned to use the troubleshooting facilities of APL, the easiest way to recover from an error is to type → to clear the workspace of terminated operators, correct the error and try again.

A common error which is often encountered is a workspace full error. It is denoted by the computer message

WS FULL

This error occurs when the user attempts to use more storage

than what is available in the active workspace. The problem can sometimes be corrected by typing → with a carriage return and then erasing all unnecessary variables and functions in the active workspace to clear it up for the problem at hand.

For more detailed information on the use of the APL language, the reader is referred to the references [54], [55].

A.4 NETWORK DESCRIPTION IN CADNAP, THE *PNET* OPERATOR

The first CADNAP operator which will be considered is the *PNET* operator. This operator is basically an analysis operator in that it takes a network and performs an analysis function without altering or modifying the network in any way. *PNET* prints out a complete description of a network on the terminal. From this description the user can completely reconstruct the network in flowgraph form.

Networks are stored in the computer in the form of vector variables whose names are assigned by the user. *PNET* is a monadic operator which does not return a result. The printout of the network is the function performed by the *PNET* operator but it cannot be assigned to another variable.

To use the *PNET* operator to print a description of a network, say *NETA*, type the instruction:

PNET NETA

The format of the printout is in the form of a listing of input and output nodes and a listing of branches. The branch

listing will contain the type of branch (*C* = coefficient branch, *D* = coefficient & delay branch), the nodes that the branch comes from and goes to, and the value of the coefficient associated with that branch. An example of a simple network and its CADNAP description by PNET are given in Fig. A.4.

Additional features of the network printout scheme will be described as necessary.

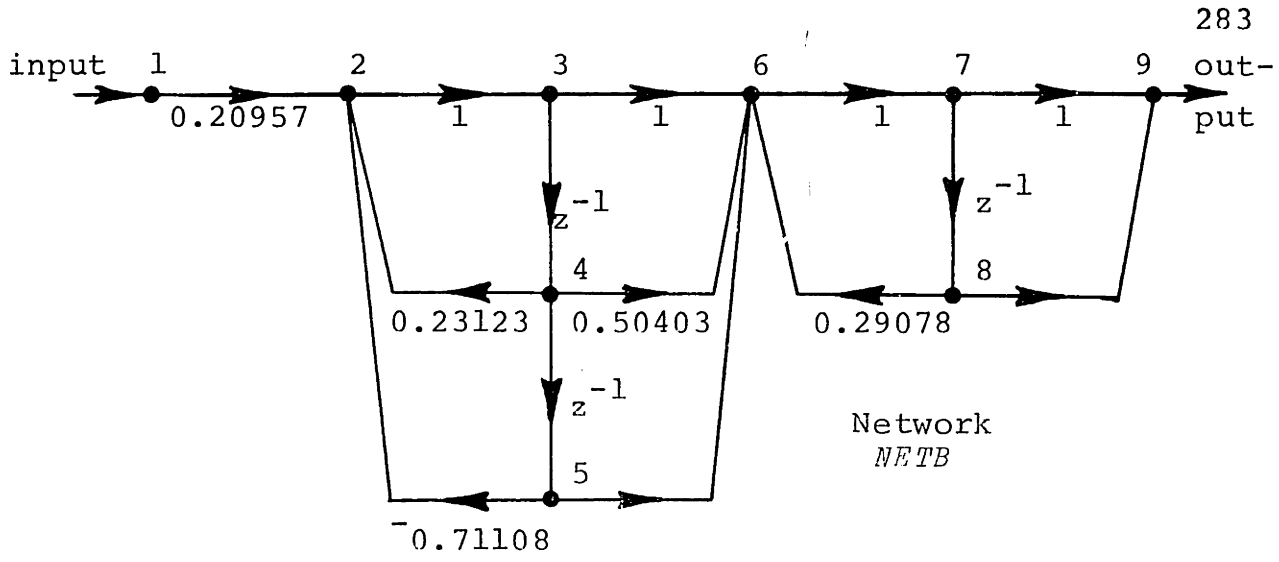
A.5 BASIC NETWORK EDITING FEATURES FOR CREATING AND CHANGING NETWORKS IN CADNAP

A.5.1 Branch Operators *BRC* and *BRD*

Two basic operators are used to insert, delete or modify branches in a network. They are dyadic operators which take the network they are modifying as the right argument and the branch description (exit node, entrance node, coefficient) as their left argument. They return a variable which is the modified network. The *BRC* operator is used to manipulate only branches with simple coefficients and the *BRD* operator is used to manipulate only branches with coefficients and unit delays. For example, to <u>insert</u> a branch consisting of a simple coefficient of value 0.756 from node 1 to node 2 in *NETA* we can do the following:

```
NETA←1 2 0.756 BRC NETA
```

Note that 1 2 0.756 is a three element vector which is taken as the left argument of *BRC* (exit node, entrance node,



CADNAP Description of *NETB*:

```
      PNET NETB

INPUT(S):  1
OUTPUT(S):  9

BRANCHES
        | TYPE | FROM  | TO | COEFFICIENT
[1]       C        1       2       0.20957
[2]       C        2       3       1
[3]       D        3       4       1
[4]       C        3       6       1
[5]       C        4       2       0.23123
[6]       D        4       5       1
[7]       C        4       6       0.50403
[8]       C        5       2      -0.71108
[9]       C        5       6       1
[10]      C        6       7       1
[11]      D        7       8       1
[12]      C        7       9       1
[13]      C        8       6       0.29078
[14]      C        8       9       1
```

Fig. A.4 A network example and its CADNAP description.

coefficient). Another point to be made is that node numbers are always restricted to be positive integers in the range 1 to 199. If a branch consisting of a simple delay already exists from node 1 to node 2 in *NETA* in the example above then *BRC* will change the value of its coefficient to the new value of 0.756. In other words, CADNAP allows only one branch of a kind (coefficient or coefficient and delay) to exist from one node to another.

The *BRC* and *BRD* operators also permit us to delete branches by assigning a coefficient of zero to that branch. For example, if there exists a branch in *NETA* from node 3 to node 4, which consists of a coefficient and delay (1 is considered as a coefficient), which we want to remove from the network, this can be done in the following manner.

```
NETA←3 4 0 BRD NETA
```

If, in the above example, there is also a branch from node 3 to node 4 consisting of a simple coefficient, this branch (simple coefficient) will not be altered.

Another feature of the branch operators *BRC* and *BRD* is that they can be used to initiate the definition of a new network by inserting branches into an empty network (empty vector) ⍳0.

```
NETB←1 2 0.35 BRC ⍳0
```

An example of the use of *BRC* and *BRD* in editing networks

will be given in A.5.2.

A.5.2 The *INPUT* and *OUTPUT* I/O Operators

The *INPUT* and *OUTPUT* operators allow the user to specify which nodes in the network are input nodes and which nodes are output nodes. They are dyadic operators and are used in the form:

NETA←S INPUT NETA

or:

NETA←S OUTPUT NETA

where *S* represents a scalar or vector of integers corresponding to the node numbers which are to be assigned as input or output nodes of the network, *NETA*.

If one or more nodes are already specified as input or output nodes, these specifications will be deleted and the new set of nodes given in *S* will be used as the input or output nodes. If *S* is the empty vector the I/O operators will delete the specifications of all input or output nodes and return a network in which no inputs or outputs are specified. Note also that networks with multiple inputs and multiple outputs are allowed in CADNAP.

Using the branch operators, I/O operators, and the cascading feature of APL operators, the network in Fig. A.4 can be easily generated. To insert the branches, the following instructions can be typed.

```
NETB←1 2 0.20957 BRC 2 3 1 BRC 3 6 1 BRC 3 4 1 BRD ι0
NETB←4 5 1 BRD 4 2 0.23123 BRC 5 2 ¯0.71108 BRC NETB
NETB←4 6 0.50403 BRC 5 6 1 BRC 6 7 1 BRC 7 9 1 BRC NETB
NETB←7 8 1 BRD 8 9 1 BRC 8 6 0.29078 BRC NETB
```

The inputs and outputs of the network can be specified by the instructions:

```
NETB←1 INPUT 9 OUTPUT NETB
```

This completes the definition of *NETB* in CADNAP. A printout of *NETB* would appear as in Fig. A.4.

A.5.3 Skeleton Branches *BRCS* and *BRDS*, Skeleton Topologies, and the Skeleton Operator *SKL*

Another editing feature of CADNAP is the capability of building up network topologies with unsepcified coefficients (skeleton networks). The coefficients can then be inserted into the skeleton network at a later time. This feature is useful when the user wants to define several networks or modules whose topologies are identical but contain different coefficients in one or more branches.

Skeleton branches (branches with unspecified coefficients) can be defined in a network with the *BRCS* and *BRDS* branch operators. They are used in the same manner as the *BRC* and *BRD* operators with the exception that, in the place of the coefficient value of the branch, we must insert a positive integer number (from 1 to 199). This integer is used to

specify where (which element) in a coefficient vector to look for the desired coefficient when we want to later assign values to the unspecified coefficients of our skeleton network. The *BRCS* and *BRDS* operators can be used to insert new branches in a network, change ordinary branches to skeleton branches if the ordinary branches already exist in the network, or change the value of the integer associated with the skeleton branch. They cannot be used to delete branches; a zero integer will result in an error message.

The skeleton operator, *SKL*, is used to insert coefficients into skeleton networks. It is a dyadic operator which takes the format

```
NETA←R SKL NET
```

where *R* is a vector of coefficients. *SKL* searches the network *NET* for skeleton branches. If the location index associated with a skeleton branch is within the range (number of elements) of the coefficient vector, *SKL* assigns the corresponding coefficient value to the branch. That is, if the location index is 3 *SKL* assigns to the branch the coefficient value of the third element in the coefficient vector *R* and converts the branch to an ordinary branch.

To demonstrate the use of *BRCS*, *BRDS*, and *SKL*, the network example of Fig. A.4 will be used. The coefficient branches from nodes 1 to 2, 4 to 2, 5 to 2, 4 to 6, and 8 to 6 can be converted to skeleton branches with associated integers 1, 2,

3, 4, and 5 respectively by typing the instruction

```
NETC←1 2 1 BRCS 4 2 2 BRCS 5 2 3 BRCS NETB
NETC←4 6 4 BRCS 8 6 5 BRCS NETC
```

In this instruction a new network, *NETC*, is defined leaving the original network *NETB* unaltered. Alternatively, if the assignment *NETB*← is made the original *NETB* would be erased and replaced by the modified network. This property is common to all of the editing operators.

A printout of *NETC* would appear as in Fig. A.5. Note that the skeleton branches are distinguished from the ordinary branches by the notation *SKL*[·] in place of the coefficient. The number in the brackets is the location index for that branch.

The original network, *NETB*, can be obtained from the skeleton network, *NETC*, and a coefficient vector by typing the instruction

```
NETB←0.20957 0.23123 ¯0.71108 0.50403 0.29078 SKL NETC
```

or by the instructions

```
R←0.20957 0.23123 ¯0.71108 0.50403 0.29078
NETB←R SKL NETC
```

A printout would look the same as that in Fig. A.4

Alternatively, any other network with the topology of *NETC* and a different set of coefficients can easily be generated by the above format. For example, the coefficient vector, *R*,

PNET NETC

INPUT(S): 1
OUTPUT(S): 9

BRANCHES

	TYPE	*FROM*	*TO*	*COEFFICIENT*
[1]	*C*	1	2	*SKL*[1]
[2]	*C*	2	3	1
[3]	*D*	3	4	1
[4]	*C*	3	6	1
[5]	*C*	4	2	*SKL*[2]
[6]	*D*	4	5	1
[7]	*C*	4	6	*SKL*[4]
[8]	*C*	5	2	*SKL*[3]
[9]	*C*	5	6	1
[10]	*C*	6	7	1
[11]	*D*	7	8	1
[12]	*C*	7	9	1
[13]	*C*	8	6	*SKL*[5]
[14]	*C*	8	9	1

Fig. A.5 Network printout of *NETC*.

PNET NETB

INPUT(S): 1
OUTPUT(S): 9

BRANCHES

	TYPE	*FROM*	*TO*	*COEFFICIENT*
[1]*	*C*	1	2	0.20957
[2]	*C*	2	3	1
[3]	*D*	3	4	1
[4]	*C*	3	6	1
[5]*	*C*	4	2	0.23123
[6]	*D*	4	5	1
[7]*	*C*	4	6	0.50403
[8]*	*C*	5	2	-0.71108
[9]	*C*	5	6	1
[10]	*C*	6	7	1
[11]	*D*	7	8	1
[12]	*C*	7	9	1
[13]*	*C*	8	6	0.29078
[14]	*C*	8	9	1

Fig. A.6 An Example of asterisk branches.

might be obtained from a synthesis program in which different sets of coefficients are inserted into a network to be analyzed. The *SKL* operator simplifies the problem by saving the user the trouble of reconstructing the network topology each time that he wants to try a new set of coefficients.

A.6 SPECIAL EDITING OPERATORS IN CADNAP

A.6.1 The *TRANSPOSE* Operator

The *TRANSPOSE* operator is a monadic operator which takes a network as its right argument and returns the transpose of the network. That is, it reverses the direction of all of the branches and interchanges inputs for outputs. It is used by the instruction format:

TRNET←TRANSPOSE NET

where *NET* is the original network and *TRNET* is the transpose network.

A.6.2 Asterisk Branches and Operators *BRCA* and *BRDA*

Asterisk branches are branches in a CADNAP structure which we would like to label or single out for special consideration in some of the analysis operations. For example, the user may want to consider only certain branches when analyzing coefficient sensitivities in a particular network (those branches in which the non-unity coefficients are located). This is done by attaching a special tag or asterisk to those

branches of interest. The operation is done by the *BRCA* or *BRDA* operators.

The *BRCA* operator applies only to coefficient branches and the *BRDA* operator applies only to coefficient and delay branches. Both operators are dyadic operators which take the network as their right argument. The left argument consists of a three element vector whose elements correspond respectively to the node number from which the branch of interest is coming (exit node), the node number to which the branch is going (entrance node), and a 1 or 0. If we want to attach an asterisk to the branch, we make the third element 1 and if we want to remove an asterisk from the branch we make it 0.

For example, in the network of Fig. A.4, if we want to attach an asterisk to those branches with non-unity coefficients, we can do so by the instructions

```
NETB←1 2 1 BRCA 4 2 1 BRCA 4 6 1 BRCA NETB
NETB←5 2 1 BRCA 8 6 1 BRCA NETB
```

A printout of the network will appear as in Fig. A.6. Note that the branches which have been labeled by the above instructions are indicated by an asterisk in the printout, consequently the term asterisk branches.

A.6.3 The Quantize Operators *QRFIX* and *QTFIX*

The quantize operators *QRFIX* and *QTFIX* enable the user to quantize coefficients in a network or in a vector or array of numbers. Quantization is done in fixed-point arithmetic.

QRFIX quantizes numbers by rounding them to their nearest binary representations and *QTFIX* quantizes them by truncation of their sign and magnitude binary representations.

The use of the quantize operators requires the presence of a global variable (scalar) in the active workspace called *MAXBIT*. This number determines the power of 2 represented by the most significant bit in the fixed-point binary number representation. For example, if the maximum bit in the representation is to be 2^0, *MAXBIT* must be set to zero. If it is to be 2^3, *MAXBIT* must be set to 3.

A second global variable which must exist in the active workspace when using the quantize operators is a variable called *EXCLUDE*. This variable is a vector of numbers which we may wish to exclude from being quantized in the coefficient vector or the network. For example if a network has the coefficients 1 and 0.533 among its coefficients and we want to exclude these two values from being quantized, we can make the assignment

```
EXCLUDE←1 0.533
```

prior to using the quantize operators. If we do not want any values to be excluded from quantization we can assign *EXCLUDE* to be the empty vector.

The quantize operators are used by the instruction format

```
NET←B QRFIX NET
```

where *B* is a scalar which tells us how many bits to quantize the coefficients to. The extra bit needed to represent the sign of a number is not counted in *B* (see Fig. A.7). If *NET* is a network, *QRFIX* (or *QTFIX*) will return a network with quantized coefficients. If *NET* is a vector or array of coefficients, it will return a vector or array of the same size with quantized coefficients. The coefficients which are excluded will be returned as well but will not be in quantized form.

In using the exclude feature it is important that the elements in *EXCLUDE* are exactly equal to the coefficients which are to be excluded. This can easily be achieved for vectors or arrays by using the index features of APL. For example if the user wants to exclude the 4th and 8th element in a coefficient vector, *CVECT*; this can be accomplished by the instruction

EXCLUDE←*CVECT*[4 8]

To obtain exact coefficient values of a network the *COEFF* and *DCOEFF* operators in Section A.7.1 can be used.

A.7 ANALYSIS FEATURES OF CADNAP

A.7.1 The *COEFF* and *DCOEFF* Operators

The *COEFF* and *DCOEFF* operators are used for obtaining the exact values of coefficients in a structure. They are classified under analysis operators because they perform an analysis

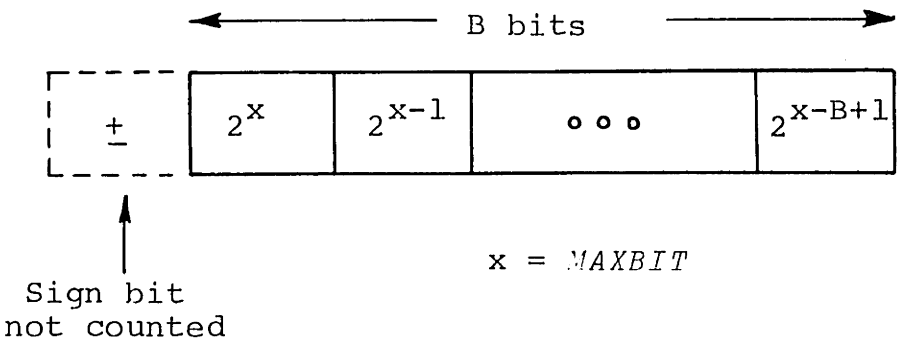


Fig. A.7 Sign and magnitude representation of a B bit number.

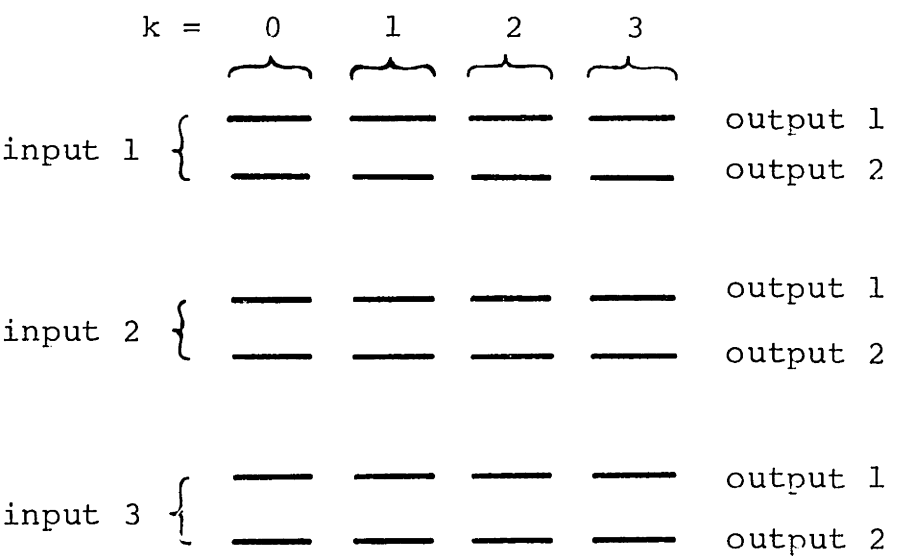


Fig. A.8 Format of an array response by *IMPULSE* for multiple input multiple output networks.

function on a network without modifying or altering it in any way. They are particularly useful in conjunction with the exclude features of the quantize operators. *COEFF* applies only to coefficient branches and *DCOEFF* applies only to coefficient and delay branches. They are dyadic operators which require a two element vector consisting of the exit node and entrance node numbers for their left argument and a network for their right argument. For example, to obtain the coefficient value of a coefficient branch from node 5 to node 3 of some network *NETA* and assign the value to a variable *C* we can type the instruction

```
C←5 1 COEFF NETA
```

A.7.2 The Impulse Response Operator, *IMPULSE*

The *IMPULSE* operator is used to generate the impulse response (unit-sample response) of a network from the input to the output. It is a monadic operator which takes a network as its right argument and returns a vector or array corresponding to the impulse response. For a network with a single input and a single output this response will be in the form of a vector. The first element of the vector corresponds to the impulse response at discrete time k = 0, the second element corresponds to k = 1 and so forth. The length of the impulse response and consequently the length of the vector is determined by a global scalar variable *DURATION* which must be present in the active workspace when using *IMPULSE*. For example, if we

want to obtain the first five terms of an impulse response we must make the assignment

DURATION←5

We can then determine the impulse response of a network *NET* and assign it to a variable *RESPONSE* by the instruction format

RESPONSE←*IMPULSE NET*

If the network *NET* has only a single input and a single output, *RESPONSE* will be a five element vector corresponding to the first five terms of the impulse response (k = 0, 1, 2, 3, 4).

For the case of multiple input, multiple output networks some clarification as to the identification of inputs and outputs is necessary. If a network is assigned to have a vector of inputs by the *INPUT* operator, then the first input will be defined to be that corresponding to the first element of this vector; the second input will correspond to the second element and so forth. For example, if a network is assigned by *INPUT* to have inputs at nodes 5 1 3, the first input is at node 5, the second is at node 1 and the third is at node 3. A similar definition applies to the outputs.

For networks with multiple inputs and/or multiple outputs *IMPULSE* returns a three dimensional array. The first plane of the array corresponds to impulse responses associated with the first input of the network; the second plane is associated with

the second input and so forth. The first row in each plane is associated with the first network output, the second row with the second output, etc. The first column corresponds to the discrete time k = 0, the second column to k = 1, etc. Consequently a network which has, for example, 3 inputs and 2 outputs and *DURATION* is 4, *IMPULSE* will return an array of numbers with 3 planes, 2 rows, and 4 columns. If we request the size of the array using the primative operator ρ, (see A.3.5) we will get the answer 3 2 4. If we want the impulse response from the second input to the first output we must look at the first row of the second plane of the array. A description of the format of the array in this example is given in Fig. A.8.

The technique used for evaluating the impulse response is that suggested by equations (5.5) and (5.6) in Chapter 5.

A.7.3 The Frequency Response Operator, *FREQ*

The frequency response operator, *FREQ*, is used to compute the frequency response of a network from input(s) to output(s). It requires the presence of a global variable in the active workspace called *FREQUENCY*. This variable must be a scalar or a vector which contains all of the frequencies at which we want to analyze the network. These frequencies must be specified in terms of radians normalized by π. Thus a frequency of 1.000 corresponds to π (z = -1 on the unit circle or one-half of the sampling frequency).

FREQ is a dyadic operator which takes a network as its

right argument and a character vector (see A.3.3) as its left argument. It is used by the instruction format

RESPONSE←A FREQ NET

where *A* represents the character vector. The character vector is used to specify the type of frequency response information that we want for the output; that is, the magnitude in dB, magnitude, phase (degrees), real part, and/or imaginary part of the frequency response. The character vector can contain one or more of the characters *L*, *M*, *P*, *R*, or *I*. These characters correspond to the operations

L ⇒ 20x log of the magnitude (i.e. dB)

M ⇒ magnitude

P ⇒ phase (degrees)

R ⇒ real part

I ⇒ imaginary part

Recall from A.3.3 that the character vector must be enclosed in quotes. Thus if we want to obtain the magnitude of the frequency response of some network *NET* and assign it to a variable *RESPONSE* we can use the instruction format

RESPONSE←'M' FREQ NET

If we want the magnitude in dB., phase, and real part we must type

RESPONSE←'LPR' FREQ NET

The output of *FREQ* for the case of a single input, single output network will be in the form of a vector. If the character vector *A* contains only one character *RESPONSE* will be a vector of the same length N as *FREQUENCY* and its elements will represent the respective frequency responses for the frequencies in *FREQUENCY*. If two characters, for example *'MP'*, appear in the character vector *RESPONSE* will be a vector of length 2N with the first N elements corresponding to the magnitude of the frequency response and the second N elements corresponding to the phase. *FREQ* always gives the outputs in the order magnitude in dB., magnitude, phase, real part, and imaginary part regardless of the order of the characters *'LMPRI'* in the character vector. For example, if we type

```
'IP' FREQ NET
```

the first N elements of the response will correspond to the phase response and the second N elements will correspond to the imaginary part.

For networks with multiple inputs and/or multiple outputs *FREQ* returns a three dimensional array in the same manner as the *IMPULSE* operator. The planes of the array are associated with network inputs and the rows in each plane are associated with network outputs (see A.7.2). Columns are associated with frequencies in the same manner as the one input one output case.

The technique used for evaluating the frequency response is that suggested by equations (5.13), (5.14), and (5.15) in

A.7.4 The Sensitivity Analysis Operators *SENS* and *SENS2*

The *SENS* operator is used to analyze sensitivities (first order derivatives) of the transfer function or system function of a network with respect to its coefficients. For single input, single output networks the transfer function from the input node to the output node is analyzed. For multiple input, multiple output networks only the transfer function from the first input to the first output (see A.7.2) is considered in the analysis. The *SENS* operator requires the vector *FREQUENCY* (see A.7.3) to be present in the active workspace to specify the frequencies at which the sensitivities must be computed.

The *SENS* operator is a dyadic operator which requires the network as its right argument and a character vector (see A.3.3 and A.3.7) as its left argument. The character vector may contain one or more of the characters *R*, *I*, *M*, *L*, and *T*. If F represents the transfer function of the network and c represents a coefficient, the characters correspond to the operations:

$$R \Rightarrow \mathrm{R_e}\ \frac{\partial F}{\partial c}$$

$$I \Rightarrow \mathrm{I_m}\ \frac{\partial F}{\partial c}$$

$$M \Rightarrow \frac{\partial |F|}{\partial c}$$

$$L \Rightarrow \frac{\partial\ \ln\ |F|}{\partial c} = \frac{1}{|F|}\ \frac{\partial |F|}{\partial c}$$

$T \Rightarrow |F|$

The output of *SENS* is a three dimensional array. Each plane corresponds to the sensitivity information with respect to one coefficient. Each row corresponds to sensitivity information associated with *R*, *I*, *M*, *L*, or *T* depending on which characters are present in the character vector. The sensitivity information corresponding to the rows is always in the order *R*, *I*, *M*, *L*, *T* independent of the ordering of the characters in the character vector. Columns in the output of *SENS* correspond to frequencies. An illustration of the format of this output is given in Fig. A.9.

The *SENS* operator analyzes the network transfer function only with respect to those coefficients associated with asterisk branches (see A.6.2). These branches must be specified by the user with the aid of the *BRCA* and *BRDA* operators. The order in which the coefficients are considered in the output array of *SENS* (see Fig. A.9) exactly corresponds to the order in which the respective asterisk branches appear in the network printout by the *PNET* operator. Consequently in Fig. A.9, coefficient 1 can be identified as the coefficient associated with the first asterisk branch that appears in the network printout, coefficient 2 corresponds to the next asterisk branch and so forth. The number of coefficients considered by *SENS* corresponds to the number of asterisk branches in the network.

The *SENS2* operator is a more specific sensitivity analysis

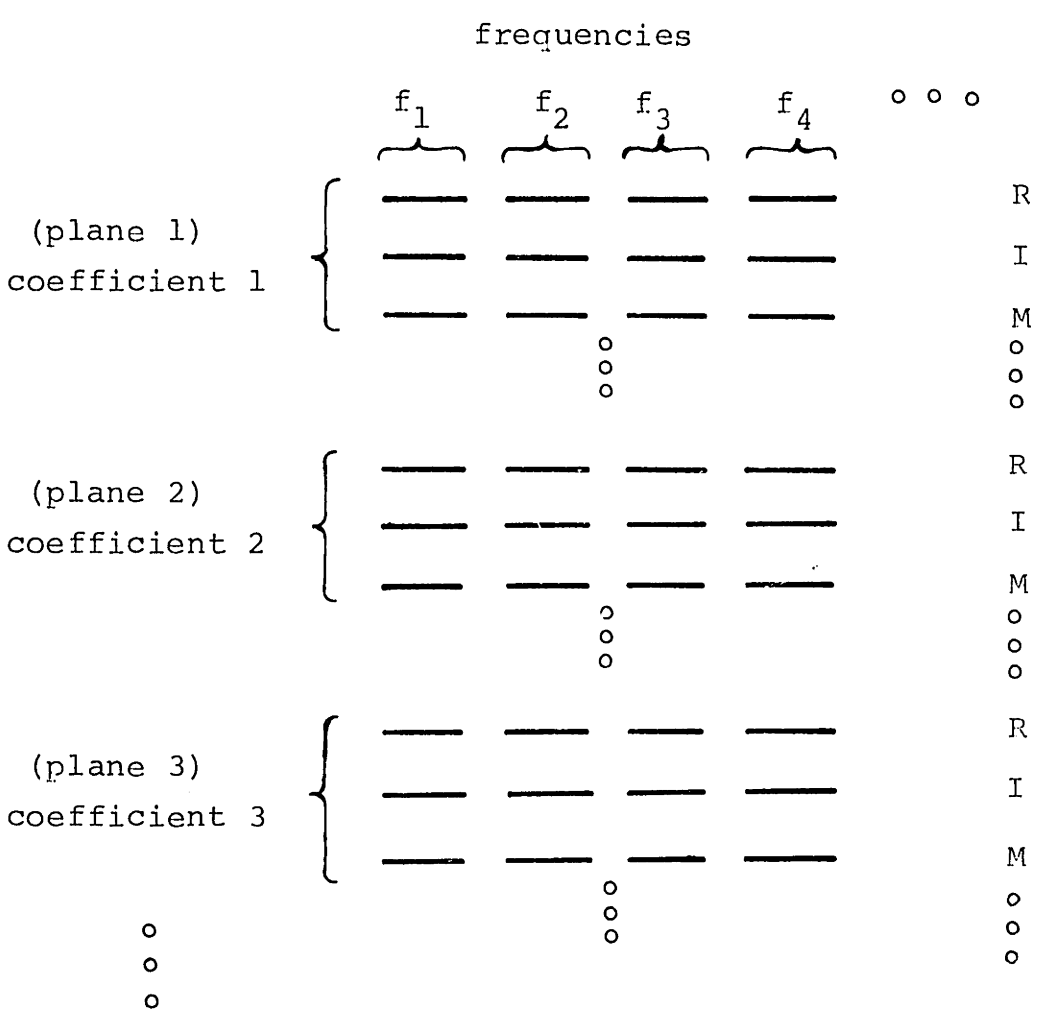


Fig. A.9 Array format for the output of the *SENS* operator.

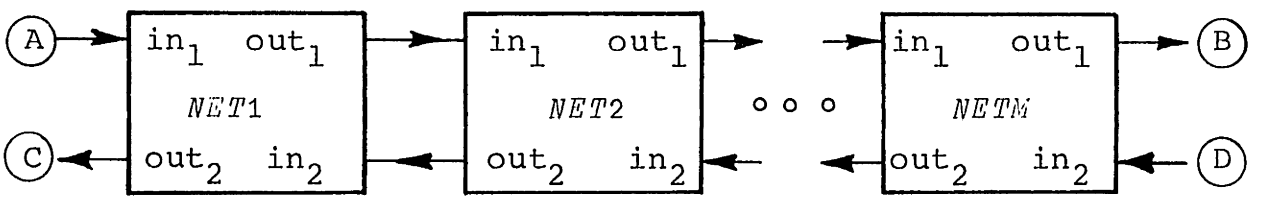


Fig. A.10 Configuration of a ladder network.

operator. It computes the sum of the squares of the sensitivities of the magnitude of the system function with respect to all of the coefficients of the asterisk branches in the network. That is, it computes the norm function

$$G = \sum_{\text{all } c=c^*} \left(\frac{\partial |F|}{\partial c} \right)^2 .$$

SENS2 is a monadic operator which takes a network as its right argument. It is used by typing the instruction

RESULT←SENS2 NET

It also requires the presence of the *FREQUENCY* vector in the active workspace. The output of *SENS2* is in the form of a 2 row, N column matrix where N is the number of frequencies in *FREQUENCY*. Consequently the columns in the output correspond to the respective frequencies in *FREQUENCY*. The first row of the matrix contains the sensitivity norms G as defined above. The second row corresponds to the magnitude of the system function |F| at the respective frequencies.

The technique used for evaluating the sensitivities is that suggested in Chapter 5, Section 5.4, and by equations (5.43) through (5.47).

A.7.5 The *LADFREQ* Operator for Analyzing the Frequency Response of Ladder Networks

For networks which are very large (greater than 20 nodes for a 32 kbyte workspace), the frequency analysis operator may

require more storage for intermediate computations than what is available in the active workspace. Attempts to analyze such networks will result in *WS FULL* errors (see A.3.7). This situation often happens in the case of ladder networks. To get around the problem the *LADFREQ* operator can be used.

To use the *LADFREQ* operator, the ladder network must be broken down into a series of smaller two input, two output subnetworks which can be connected in the manner depicted in Fig. A.10. Each subnetwork *NET*1 through *NETM* must be separately defined by the network editing operators and stored in the active workspace. *LADFREQ* connects these networks in the manner depicted in Fig. A.10 and analyzes the overall network by analyzing the subnetworks one at a time.

LADFREQ is used by the instruction format:

RESPONSE←*A LADFREQ NET*1,*NET*2,...,*NETM*

The left argument must contain the subnetworks, separated by commas, in the order that they appear in the network. The right argument is a character vector. This vector must contain one or more of the characters *L*, *M*, *P*, *R*, or *I* which correspond to the same operations as in the *FREQ* operator. In addition the character vector must contain one or more of the numbers 1, 2, 3, or 4. These numbers are used to identify which transfer functions are to be computed. The transfer functions can be identified with the aid of Fig. A.10 as

1 transfer function from A to B = F_{AB}

2 transfer function from A to C = F_{AC}

3 transfer function from D to C = F_{DC}

4 transfer function from D to B = F_{DB}

If only one of the above numbers is present in the character vector the output of *LADFREQ* will be a vector and will have the same format as the output of *FREQ* for one input, one output networks. If more than one of the numbers 1, 2, 3, or 4 is present the output will be in the form of a matrix with each row corresponding to one of the transfer functions F_{AB}, F_{AC}, F_{DC}, or F_{DB}. The transfer functions which appear depends on the choice of the above numbers. They will always appear in the order given above independent of the order of the numbers in the character vector. The columns will again have the same correspondence as they do in the output of the *FREQ* operator.

The *LADFREQ* operator can also be used to analyze cascades of single input, single output networks. This can be accomplished by assigning the second inputs and outputs of each of the subnetworks to isolated nodes. The forward and reverse paths in the overall structure are then decoupled and the transfer function of the cascaded network is simply F_{AB}.

A.8 PLOTTING FACILITIES FOR CADNAP

The plotting package in CADNAP has been borrowed from the public workspace *PLOTFORMAT* (or *PLOTFMT*) [55]. It consists of

three functions *PLOT*, *AND*, and *VS*. To use the *PLOT* program to plot a vector *A* against a vector *B* we can type

SCALESIZE PLOT A VS B

where *SCALESIZE* is a scalar or a two element vector which determines the approximate size of the plot in character spaces. If we want to plot two vectors *A* and *C* simultaneously against a vector *B* we can type

SCALESIZE PLOT A AND C VS B

where vectors *A*, *B*, and *C* must be of the same length.

If *SCALESIZE* is a scalar, say 50, we will obtain a plot approximately 50 character spaces high and 50 character spaces wide. If it consists of two numbers, say 30 40, we will obtain a plot approximately 30 character spaces high and 40 character spaces wide. It is important to keep in mind when choosing *SCALESIZE* that the height of a character is about 1.66 times its width. Consequently, choosing one number for *SCALESIZE* or two numbers of the same value does not produce a square plot. For further information on the details of *PLOT* the reader is referred to reference [55].

As an example of the use of *PLOT* the frequency response of the network in Fig. A.4 is plotted for 50 frequency points which are equally spaced from 0 to 0.94π. To generate the *FREQUENCY* vector we can type the instruction

FREQUENCY←0.94×(0,ι49)÷49

The magnitude of the frequency response in dB. can then be determined and assigned to the vector *GAIN* by typing

```
GAIN←'L' FREQ NETB
```

To plot this response we can type the instruction

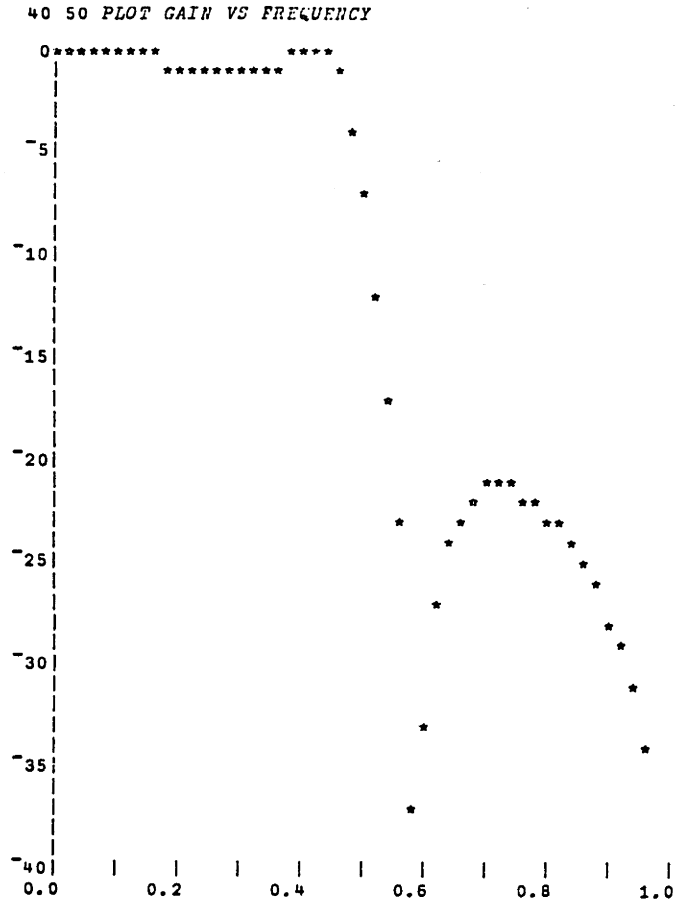
```
40 50 PLOT GAIN VS FREQUENCY
```

This plot is given in Fig. A.11(a). The filter response corresponds to a third-order lowpass elliptic filter with a 1 dB. ripple in the passband and a 21 dB. attenuation in the stopband.

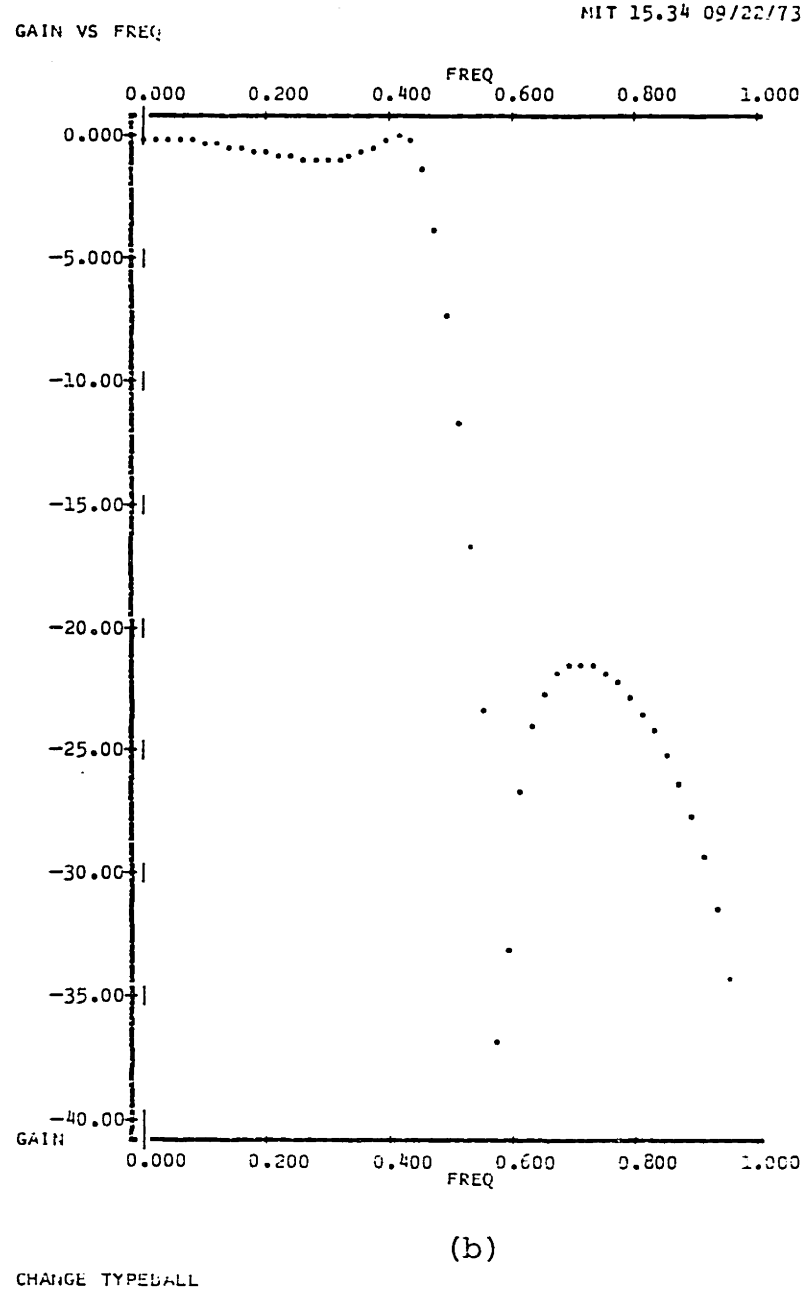
The plotting resolution of *PLOT* is limited by the size of a character space. If the plot is 50 character spaces wide then the horizontal resolution is 1/50 of the full width of the plot. If greater resolution is required other types of plotting routines are available in APL which require the use of a special high resolution plotting typeball. One such routine is *ACCUPLOT*. An example of a plot using this plotting routine is given in Fig. A.11(b). This routine is not stored in the CADNAP workspace but it or a similar program may be available in the public files of many APL systems.

A.9 ORGANIZATION AND USE OF THE CADNAP WORKSPACE

All of the editing and analysis programs described in A.4, A.5, A.6, A.7 and A.8 are located in the workspace called CADNAP. Together these programs consume about 23 kbytes of



(a)



(b)

Fig. A.11 A plot of the frequency response of *NETB* using (a) *PLOT* and (b) *ACCUPLOT* with a high resolution typeball (60% scale reduction on both plots).

storage. For a 32 kbyte (1 kbyte = 1000 bytes) workspace, which is standard for many APL systems, this amounts to about 72 per cent of the workspace. The remaining storage space, about 9 kbytes, is available for storage of variables and temporary storage of data during computation. If this is not a sufficient amount of storage for a given problem, some of the operators can be erased from the active workspace to provide more storage. This can be done because the operators in CADNAP are all independent of each other. For example, if we want to compute the frequency response of a network, the only things that we need to have in the active workspace are the network, the *FREQUENCY* vector and the *FREQ* operator. There are two exceptions to the above rule; the branch operators *BRC*, *BRD*, *BRCS*, and *BRDS* each require the presence of the *BR* operator and the *LADFREQ* operator requires the presence of the *FREQ* operator in the active workspace.

To simplify the process of erasing or copying programs into or out of the active workspace, programs are organized into groups. Using this feature, groups of programs can be transferred into or out of the active workspace simultaneously with a single *COPY* or *ERASE* command using the group name. Table A-1 gives a listing of these groups and the number of bytes necessary to store them.

When analyzing large networks there are several things that we can do to avoid *WS FULL* errors. One is to keep only the necessary operators in the active workspace as discussed

Table A-1 Groups in CADNAP

Major Group	Subgroups	Operators	Bytes/operator
GEDIT (9068)	*GBRANCH* (3896)	*BR*	2632
		BRC	72
		BRD	76
	GSKL (952)	*BRCS*	76
		BRDS	76
		SKL	776
	GASTERISK (944)	*BRCA*	456
		BRDA	468
	GIO (748)	*INPUT*	376
		OUTPUT	352
	GQUANT (1012)	*QTFIX*	468
		QRFIX	524
		TRANSPOSE	564
		PNET	1424
	GCOEFF (672)	*COEFF*	324
		DCOEFF	328
		IMPULSE	1164
	GLADFREQ (3524)	*FREQ*	1756
		LADFREQ	1732
	GSENS (3904)	*SENS*	2104
		SENS2	1780
	GPLOT (4740)	*PLOT*	3256
		AND	980
		VS	480

above. In this way we can analyze networks of up to 20 nodes in size in a 32 kbyte workspace using *IMPULSE*, *FREQ*, *SENS*, or *SENS2*. Another helpful technique is to define networks with as few nodes as possible. For example, in the network in Fig. A.4 nodes 2 and 3 could be combined and nodes 6 and 7 could be combined without any loss in generality. The number of nodes in a network directly determines the size of the matrices that must be manipulated and therefore the amount of intermediate storage required for the analysis. Another way to avoid *WS FULL* errors when analyzing frequency responses for cascade or ladder networks is to use the *LADFREQ* operator. In this way the analysis of a large network is broken down into a series of smaller analyses of subnetworks. Finally, the most effective way to avoid *WS FULL* errors is to acquire larger workspaces. The user can check with the system operator of his particular system to determine if this is possible.

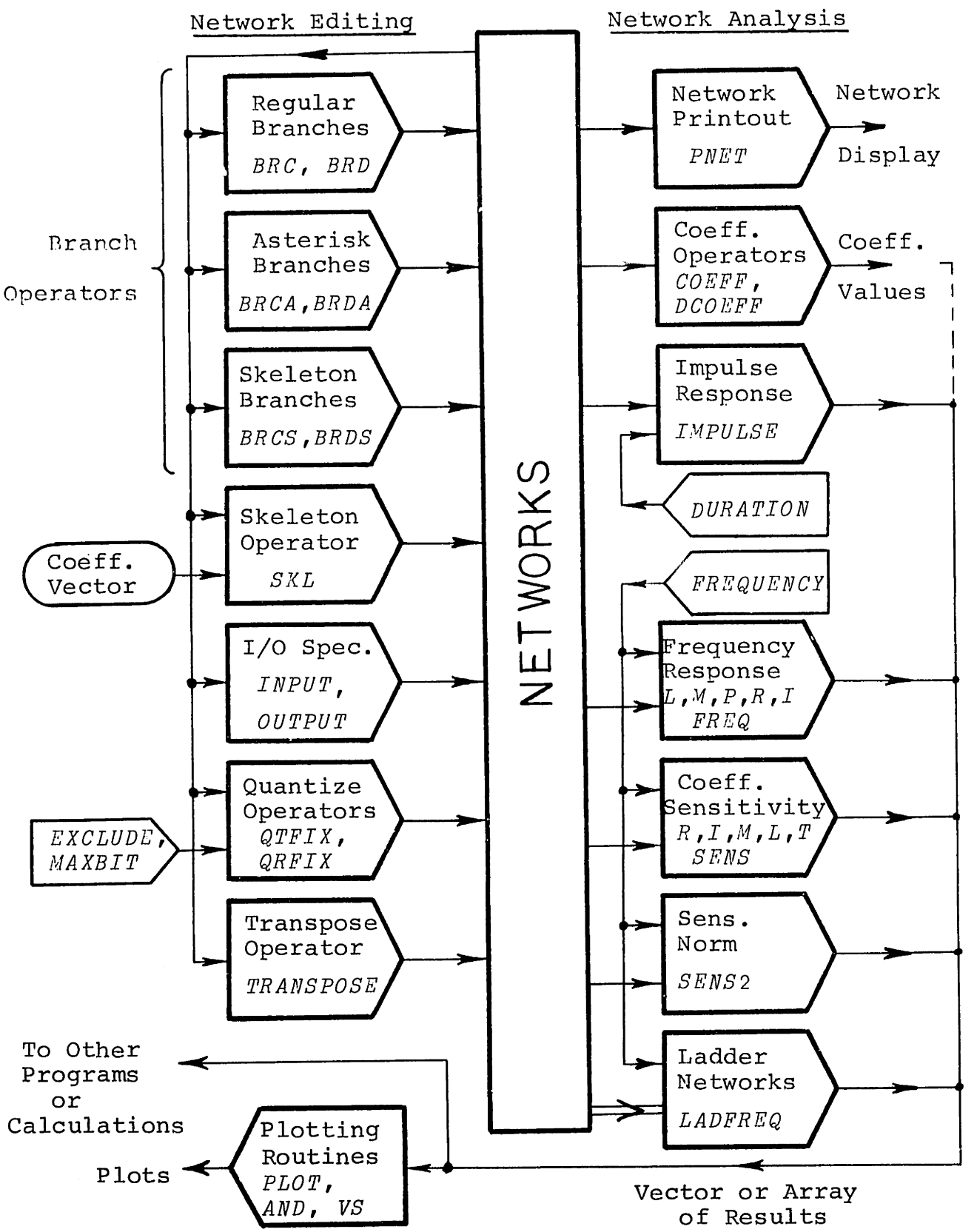
For quick reference the operators and programs in CADNAP are summarized in Table A-2 in the form of an operational chart.

A.10 THE INTERNAL COMPOSITION OF NETWORK VECTORS

Networks in CADNAP are stored compactly in the form of vector variables rather than in matrix form. In this way much less storage is required in the active workspace for storing network information. All of the CADNAP operators recognize networks only in this format.

Table A-2

OPERATIONAL CHART OF CADNAP



Information in a network vector is stored according to branch descriptions and input/output node specifications. The composition of a network vector is depicted in Fig. A.12. The double line represents the network vector and the space between each major division on this line corresponds to an element or number in the vector. The vector can be broken down into several major parts as depicted in Fig. A.12.

Element 1, the first number in the vector, corresponds to a label and is always equal to the value 987654321. This label distinguishes the network vectors from all other vectors which may be present in the workspace.

Elements 2 and 3 are pointers. They are used to identify where in the network vector to look for various pieces of information about the network. Each of these elements corresponds to a nine decimal digit integer number in which the decimal digits are taken in groups of three. Numbers A and D correspond to the three most significant decimal digits of the first and second elements respectively. B and E correspond to the next three significant decimal digits of each element respectively and C and F correspond to the least three significant decimal digits. The numbers A, B, C, D, E, and F therefore each represent positive integers between 000 and 999. These numbers correspond to element locations in the network vector which identify the location of major subdivisions of network information (see Fig. A.12). The number D is not used in the present stage of development of CADNAP. It has been included

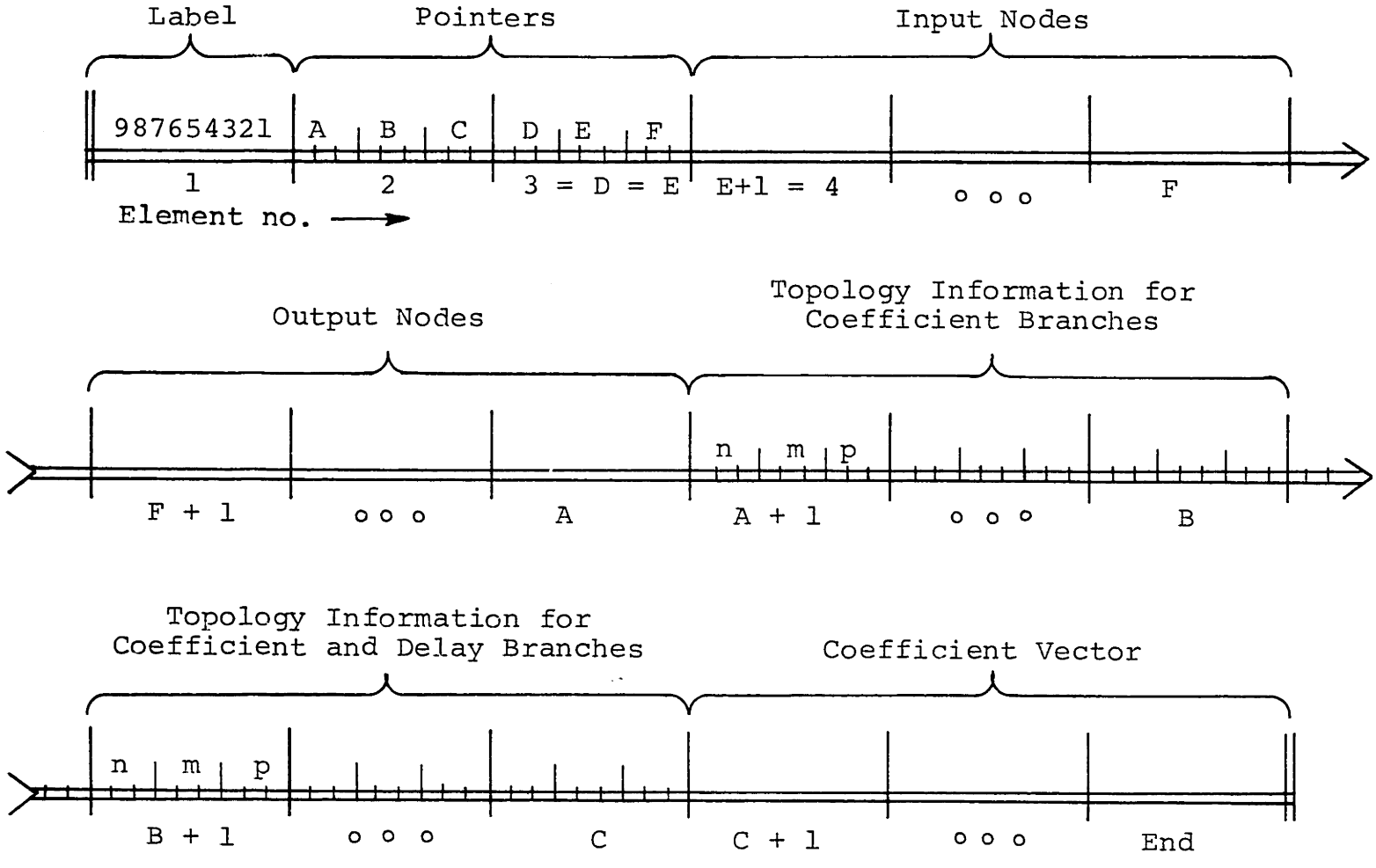


Fig. A.12 Composition of a network vector.

to allow for future expansion of the network description. The number E is set to 3 and corresponds to the element location just prior to the input node listing in the network. If new information is to be added to a network it can be added between elements 3 and 4 in the network with the value of E updated appropriately. Additions made by this format will not require any modifications to any of the present operators in CADNAP.

Elements E+1 through F correspond to input nodes of the network. One element corresponds to each node number. The first element (E+1) corresponds to the first input node, the second element corresponds to the second input and so forth. If no inputs are specified in a network, no elements will be present in this part of the vector. In this case E = F.

Elements F+1 through A correspond to output nodes of the network. The same format as that for the inputs applies here.

Elements A+1 through B correspond to topology information for coefficient branches. If A = B, no coefficient branches exist in the network. Each element in this grouping corresponds to one coefficient branch in the network and contains all of the topological information about that branch except its coefficient value. As in the case of the pointers, these elements correspond to nine decimal digit numbers in which the decimal digits are divided into groups of three decimals each. These three numbers are identified as m, n, and P, (see Fig. A.12) and each number can take on integer values between 0 and 999.

The number n corresponds to the node to which the branch is going and m corresponds to the node number from which the branch is coming. The integer P is the sum of three numbers r, s, and t; that is P = r + s + t. If the branch is a skeleton branch, r = 400, and if it is not r = 0. If the branch is an asterisk branch, s = 200, and if not, s = 0. The number t takes on a positive integer value, $1 \leq t \leq 199$, which corresponds to the element location in the coefficient vector for which to find the coefficient value of the branch. If t = 5, this implies that the coefficient value can be found in the fifth element of the coefficient vector or element C+5 in the network vector. If more than one branch contains the same coefficient value, this value will be stored only once in the coefficient vector instead of being duplicated. The value of P can now be identified as follows:

$1 \leq P \leq 199$	ordinary branch
$201 \leq P \leq 399$	asterisk branch
$401 \leq P \leq 599$	skeleton branch
$601 \leq P \leq 799$	asterisk, skeleton branch.

Elements B+1 through C correspond to coefficient and delay branches. The same format as that for the coefficient branches applies here.

Elements C+1 to the end of the network vector are identified as coefficients in the structure. If a given coefficient value appears more than once in the structure, for example a

coefficient of unity, this coefficient value will appear only once in the coefficient vector. In this way the length of the vector is kept as short as possible. Coefficients are stored in double precision floating-point format.

A.11 OTHER PROGRAMS IN APL

Besides the CADNAP programs a number of other programs for digital signal processing applications are available in APL (see Ref. [56]). Included are programs for synthesizing lowpass, highpass, and bandpass Butterworth or Tchebyschev filters and lowpass elliptic filters. Also numerous programs are also available in public files of APL for solving numerical problems such as linear programming, factoring, plotting, etc. Any of these programs could be used in combination with the CADNAP programs.

APPENDIX B

LISTING OF CADNAP PROGRAMS

```
      )FNS
AND     BR      BRC     BRCA    BRCS    BRD     BRDA    BRDS    COEFF   DCOEFF
FREQ    IMPULSE INPUT   LADFREQ OUTPUT  PLOT    PNET    QRFIX   QTFIX   SENS
SENS2   SKL     TRANSPOSE       VS
      )VARS
      )GRPS
GASTERISK       GBRANCH GCOEFF  GEDIT   GIO     GLADFREQ        GPLOT   GQUANT
GSENS   GSKL
      )GRP GASTERISK
BRCA    BRDA
      )GRP GBRANCH
BRC     BRD     BRCS    BRDS    BR      BRCA    BRDA
      )GRP GCOEFF
COEFF   DCOEFF
      )GRP GEDIT
BR      BRC     BRD     BRCS    BRDS    SKL     BRCA    BRDA    INPUT   OUTPUT
QTFIX   QRFIX   TRANSPOSE         PNET    COEFF   DCOEFF
      )GRP GIO
INPUT   OUTPUT
      )GRP GLADFREQ
FREQ    LADFREQ
      )GRP GPLOT
PLOT    VS      AND
      )GRP GQUANT
QRFIX   QTFIX
      )GRP GSENS
SENS    SENS2
      )GRP GSKL
BRCS    BRDS    SKL
```

```
      ∇BRC[□]∇
    ∇ Z←X BRC Y
[1]   Z←(1,X) BR Y
    ∇

      ∇BRD[□]∇
    ∇ Z←X BRD Y
[1]   Z←(2,X) BR Y
    ∇

      ∇BRCS[□]∇
    ∇ Z←X BRCS Y
[1]   Z←(3,X) BR Y
    ∇

      ∇BRDS[□]∇
    ∇ Z←X BRDS Y
[1]   Z←(4,X) BR Y
    ∇

      ∇BRCA[□]∇
    ∇ R←B BRCA NET;C;L
[1]   →ER×⍳∨/(NET[1]≠987654321),(~B[1 2]∈⍳199),~B[3]∈ 0 1
[2]   →ER×⍳(C←(⌊(L[1]+L[2]↑NET)÷1000)⍳1000⊥B[2 1])>L[2]-(L← 1000 1000 1000 ⊤NET[
      2])[1]
[3]   →0,R←NET,0⍴NET[C]←NET[C]+200×(B[3]-1)+200>400|1000|NET[C←L[1]+C]
[4]  ER:'IMPROPER NETWORK OR BRANCH NOT FOUND',0⍴R←NET
    ∇
```

```
      ∇ERRDA[⎕]∇
    ∇ R←B BRDA NET;C;L
[1]   →ER×ι∨/(NET[1]≠987654321),(~B[1 2]∈ι199),~B[3]∈ 0 1
[2]   →ER×ι(C←(⌊(L[2]↓L[3]↑NET)÷1000)ι1000⊥B[2 1])>L[3]-(L← 1000 1000 1000 ⊤NET[
      2])[2]
[3]   →0,R←NET,0ρNET[C]←NET[C]+200×(B[3]-1)+200>400|1000|NET[C←L[2]+C]
[4]  ER:'IMPROPER NETWORK OR BRANCH NOT FOUND',0ρR←NET
    ∇


      ∇INPUT[⎕]∇
    ∇ R←B INPUT NET;LL
[1]   →(∨/(NET[1]≠987654321),(0ρLL←(3ρ1000)⊤NET[3]),~B∈ι199)/ER
[2]   →0,R←NET[1],(1000⊥(LL[2]-LL[3]-ρ,B)+(3ρ1000)⊤NET[2]),(1000⊥LL[1 2],LL[
      2]+ρ,B),B,LL[3]↓NET
[3]  ER:'IMPROPER USE OF INPUT OPERATOR OR IMPROPER NETWORK'
    ∇


      ∇OUTPUT[⎕]∇
    ∇ R←B OUTPUT NET;LL
[1]   →(∨/(NET[1]≠987654321),~B∈ι199)/ER
[2]   →0,R←NET[1],(1000⊥R+LL[3]-R[1]-ρ,B),(2↓LL[3]↑NET),B,(1↑R←(3ρ1000)⊤NET[
      2])↓NET,0ρLL←(3ρ1000)⊤NET[3]
[3]  ER:'IMPROPER USE OF OUTPUT OPERATOR OR IMPROPER NETWORK'
    ∇
```

```
      ∇BR[□]∇
    ∇ R←B BR NET;L;H;G;PP
[1]     →(ER×⍳∨/(1 1 1 ,∨/B[4]= 3 4)∧~B∈⍳199),L1×⍳0≠⍴NET
[2]     R←987654321,(1000⊥3+0,(0≠B[4])×(∨/B[1]= 1 3)+0,∨/B[1]= 2 4),3003003
[3]     →0,R←R,((B[4]=0)↓,1000⊥B[3 2],(∨/B[1]= 1 2)+(∨/B[1]= 3 4)×400+R[4]),((B[
        4]=0)-∨/ 1 2 =B[1])↑B
[4]   L1:→(987654321≠NET[1],0⍴L←(3⍴1000)⊤NET[2])/ER
[5]     →(0=⍴H←L[(∨/B[1]= 1 3)+2×∨/B[1]= 2 4]↓L[(2×∨/B[1]= 1 3)+3×∨/P[1]= 2
        4]↑NET)/L2
[6]     →((G←(⌊H÷1000)⍳1000⊥B[3 2])>⍴H)/L2
[7]     →(0≠B[4],0⍴H←(3⍴1000)⊤NET[G←G+L[(∨/B[1]= 1 3)+2×∨/P[1]= 2 4]])/L3
[8]     →(H[3]<400,0⍴NET←((G-1)↑NET),G↓NET,0⍴NET[2]←1000⊥L←L+0,(-∨/B[1]= 1
        3),¯1)/L4
[9]   L5:→0,R←NET
[10]  L4:→(((G←(200|G)+400×400<G←1000|L[1]↓L[3]↑NET)⍳H[3]←200|H[3])≤⍴L[1]↓L[
        3]↑NET)/L5
[11]    →0,R←(L[1]↑R),((L[1]↓L[3]↑R)-(G≥H[3])∧G≤400),L[3]↓R←((L[3]+H[3]-1)↑NET),(L[
        3]+H[3])↓NET
[12]  L3:→(∨/B[1]= 1 2)/L6
[13]    →(L4×⍳H[3]<400,0⍴NET[G]←1000⊥H[1 2],400+B[4]+200×200<400|1000|H[3]),L5
[14]  L6:→(H[3]<400)/L7
[15]    →0,R←NET,((200|H[3])>(⍴NET)-L[3])↑R[4],0⍴NET[G]←1000⊥H←H[1 2],((L[3]↓NET)⍳B[
        4])+200×H[3]>600
[16]  L7:→(2≤+/((200|PP)+400×400<PP←1000|L[1]↓L[3]↑NET)=200|H[3])/L8
[17]    →L5,NET[L[3]+200|H[3]]←R[4]
[18]  L8:→L5,NET[G]←1000⊥H,0⍴NET←NET,((200|P[3]←((L[3]↓NET)⍳B[4])+200×H[3]>
        200)>⍴L[3]↓NET)↑B[4]
[19]  L2:→(B[4]=0)/L5
[20]    G←(+/(H,10*9)≤1000⊥B[3 2],900)+L[(∨/B[1]= 1 3)+2×∨/P[1]= 2 4]
[21]    NET[2]←1000⊥L←L+0,(∨/B[1]= 1 3),1,0⍴NET←(G↑NET),(1000⊥H←R[3 2],(B[4]+
        400)×∨/B[1]= 3 4),G↓NET
[22]    →(L5×⍳∨/B[1]= 3 4 ,0⍴G←G+1),L8
[23]  ER:'IMPROPER USE OF BRANCH OPERATOR'
    ∇
```

```
      ∇SKL[□]∇
    ∇ R←A SKL NET;L;H;S;G;PP
[1]   →(∨/(NET[1]≠987654321),0ρH←((200|H)≤ρA←,A)∧400<H←1000|L⌈1]+L[3]↑NFT,0ρL←(
      3ρ1000)⊤NET[2])/ER
[2]   L1:→((S←H⍳1)>ρH)/L2
[3]   →(0≠PP←A[200|G[3]],0ρG←(3ρ1000)⊤NET[L[1]+S])/L3
[4]   →L1,H←((S-1)↑H),S↓H,0ρNET←((L[1]+S-1)↑NFT),(L[1]+S)↓NET,0ρNFT[2]←1000⊥L←L-0,
      (S≤L[2]-L[1]),1
[5]   L3:→L1,H[S]←0,0ρNET←NET,(R>ρL[3]↓NFT)↑PP,0ρNFT[L[1]+S]←1000⊥(2↑G),(200×G⌈
      3]>600)+R←(L[3]↓NET)⍳PP
[6]   L2:→0,R←NET
[7]   ER:'IMPROPER USE OF SKL OPERATOR OR IMPROPER NETWORK'
    ∇


      ∇TRANSPOSE[□]∇
    ∇ R←TRANSPOSE NET;HC;HD;L;LL
[1]   →(∨/(NET[1]≠987654321),0ρL←(3ρ1000)⊤NET[2])/ER
[2]   HC←(L[2]←L[2]-L[1])↑HD←(3ρ1000)⊥((3ρ1000)⊤L[1]↓L[3]↑NFT)[2 1 3 ;]
[3]   R←(HC[⍋HC]),(HD[⍋HD←L[2]↓HD]),L[3]↓NET
[4]   →0,R←NET[1 2],(NET[3]+L[1]+LL[2]-2×LL[3]),(LL[3]↓L[1]↑NFT),(LL[2]↓(LL←(
      3ρ1000)⊤NET[3])[3]↑NET),R
[5]   ER:'NOT A PROPER NETWORK'
    ∇


      ∇COEFF[□]∇
    ∇ R←A COEFF NET;LL;T
[1]   →ER×⍳∨/(NET[1]≠987654321),2≠ρA
[2]   →ER×⍳(A←(⌊T÷1000)⍳1000⊥⍋A)>ρT←LL[1]↓(LL← 1000 1000 1000 ⊤NFT[2])[2]↑NFT
[3]   →0,R←NET[LL[3]+200|T[A]]
[4]   ER:'IMPROPER NETWORK OR BRANCH NOT FOUND'
    ∇
```

```
      ∇DCOEFF[□]∇
    ∇ R←A DCOEFF NET;LL;T
[1]   →ER×ι∨/(NET[1]≠987654321),2≠ρA
[2]   →ER×ι(A←(⌊T÷1000)ι1000⊥ϕA)>ρT←LL[2]↓(LL← 1000 1000 1000 ⊤NFT[2])[3]↑NFT
[3]   →0,R←NET[LL[3]+200|T[A]]
[4]  ER:'IMPROPER NETWORK OR BRANCH NOT FOUND'
    ∇


      ∇QTFIX[□]∇
    ∇ R←B QTFIX C;N;S;T
[1]   →(ER×ι0≠1|B),(L1×ι987654321≠1↑,C),N←0ρS←ρC
[2]   C←R↓C,0ρN←(R←1000|C[2])↑C
[3]  L1:→ER×ι∨/(2*MAXBIT+1)<|C←C-C×(T←EXCLUDE ι C)≤ρEXCLUDE←,EXCLUDE
[4]   →0,ρR←SρN,((1-2×C<0)×((Bρ2)⊥⌊(Bρ2)⊤,|C×R)÷R←2*B-MAXBIT+1)+(EXCLUDE,0)[T]
[5]  ER:'COEFFICIENT VALUE GREATER THAN 2*(MAXBIT+1)  OR B NOT INTEGER'
    ∇


      ∇QRFIX[□]∇
    ∇ R←B QRFIX C;N;S;T
[1]   →(ER×ι0≠1|B),(L1×ι987654321≠1↑C←,C),N←0ρS←ρC
[2]   C←R↓C,0ρN←(R←1000|C[2])↑C
[3]  L1:→ER×ι∨/((2*MAXBIT+1)-2*MAXBIT-B)<|C←C-C×(T←EXCLUDE ι C)≤ρEXCLUDE←,EXCLUDE
[4]   →0,ρR←SρN,((1-2×C<0)×((Bρ2)⊥⌊0.5+(Bρ2)⊤,|C×R)÷R←2*B-MAXBIT+1)+(EXCLUDE,0)[T]
[5]  ER:'COEFFICIENT VALUE GREATER THAN (2*(MAXBIT+1))-2*(MAXBIT-B), (*=POWER), OR
      B NOT INTEGER'
    ∇
```

```
      ∇PNET[□]∇
    ∇ PNET NET;L;H;N;R;X;LL;I;J;K;XX
[1]   →(∨/(3>ρNET),NET[1]≠987654321,ιN←0)/ER
[2]   0ρL←,(3ρ1000)⊤NET[2],0ρLL←(3ρ1000)⊤NET[3]
[3]   (11×0=LL[3]-LL[2])↓'INPUT(S):  ';LL[2]↓LL[3]↑NET
[4]   (12×0=L[1]-LL[3])↓'OUTPUT(S):  ';LL[3]↓L[1]↑NET
[5]   0ρX←' CDSKL[',0ρI←'[',0ρJ←']',0ρK←' ',0ρXX←' *'
[6]   'BRANCHES'
[7]   '       | TYPE | FROM  | TO | COEFFICIENT'
[8]   →(0=ρH←L[1]↓L[3]↑NET)/0
[9]   H←L[1]+⍋(1000 1000 ⊥(1000 1000 1000 ⊤H)[2 1 ;])+(L[1]↓L[2]ρ0),L[2]↓L[
      3]ρ0.5
[10] L1:R←(2↑(3ρ1000)⊤R),(200|R),(400<1000|R←NET[H[N]]),1+200<400|1000|NET[H[N←N+1
      ]]
[11]  R←R,(3++/N< 10 100),(2+H[N]>L[2]),(5++/R[2]< 10 100),(4++/R[1]< 10
      100),NET[L[3]+R[3]×~R[4]]
[12]  I;N;J;XX[R[5]];R[6]ρK;X[R[7],R[8]ρ1];R[2];R[9]ρK;R[1];6ρK;(-4×R[4])↑X;R[
      4]↑R[3];R[4]↑J;R[4]↓R[10]
[13]  →(0×ιN=ρH),L1
[14] ER:'NOT A PROPER NETWORK'
    ∇
```

```
      ∇IMPULSE[□]∇
    ∇ R←IMPULSE NET;L;ND;N;NN;IPT;OPT;B;S;YK;A;LL
[1]   →ER×⍳∨/(NET[1]≠987654321),(DURATION<1),0≠1|DURATION,0⍴L←(3⍴1000)⊤NET[
      2]
[2]   →ER×⍳∨/(400<1000|L[1]+L[3]↑NET),(0=⍴IPT←LL[2]+LL[3]↑NET),0=⍴OPT←(LL←(
      3⍴1000)⊤NET[3])[3]+L[1]↑NET
[3]   S←(⍴IPT←ND⍳IPT),(⍴OPT←ND⍳OPT),0⍴NN←N,N←⍴ND←(N∈ND)/N←⍳⌈/ND←IPT,OPT,,
      ¯1 0 ↓B←200|(3⍴1000)⊤L[1]+L[3]↑NET
[4]   ND←(ND⍳,B[2;])+N×(ND⍳,B[1;])-⍴⍴NET←(L[3]+NET)[, 2 0 ↓B]
[5]   B[L↑ND]←B[L↑ND]-(L←L[2]-L[1])↑NET,B←,NN⍴((N,1)⍴1),NN⍴0
[6]   YK[IPT-N×1-⍳⍴IPT]←⍴⍴YK←(N×⍴IPT)⍴0
[7]   YK←(B←⍉[⍳2]NN⍴B)+.×⍉((⍴IPT),N)⍴YK
[8]   A[L+ND]←L+NET,0⍴A←,NN⍴0
[9]   N←S,L←B←⍴⍴⍴A←B+.×NN⍴A
[10]  →L2×⍳DURATION=⍴⍴⍴S←N⍴⍉YK[OPT;]
[11] L1:→L1+DURATION≤L←L+⍴⍴⍴S←S,N⍴⍉(YK←A+.×YK)[OPT;]
[12] L2:→L3×∧/ 1 1 =2↑⍴R←S
[13] L3:→0,R←,S
[14] ER:'DURATION INCORRECT, INPUT/OUTPUT UNSPECIFIED, SKL BRANCH, OR IMPROPER NET
      WORK'
    ∇
```

```
      ∇FREQ[□]∇
    ∇ R←G FREQ NET;T;L;IPT;OPT;B;N;ND;NN;S;A;XX
[1]   →ER×ι∨/(NET[1]≠987654321),(∧/G←~'LMPRI'∈G),(5<ρG),0=T←ρFREQUENCY←,FREQUENCY,
      0ρL←(3ρ1000)⊤NET[2]
[2]   →ER×ι∨/(400<1000|L[1]↓L[3]↑NET),(0=ρIPT←A[2]↓A[3]↑NET),0=ρOPT←(A←(3ρ
      1000)⊤NET[3])[3]↓L[1]↑NET
[3]   S←((ρIPT←ND⍳IPT),(ρOPT←ND⍳OPT),0)ρNN←N,N←ρND←(N∈ND)/N←ι⌈/ND←IPT,OPT,,
      ¯1 0 ↓B←200|(3ρ1000)⊤L[1]↓L[3]↑NET
[4]   ND←(ND⍳,B[2;])+N×(ND⍳,B[1;])-ρρNET←(L[3]↓NET)[, 2 0 ↓B]
[5]   B[L↑ND]←B[L↑ND]-(L←L[2]-L[1])↑NET,B←,NNρ((N,1)ρ1),NNρ0
[6]   XX[IPT-N×1-ιρIPT]←ρρXX←(N×ρIPT)ρ0
[7]   A[L↓ND]←L↓NET,0ρA←,NNρ0
[8]   ND←(NNρ((N,1)ρ1),NNρ0)+A+.×A←(B←⍸NNρB)+.×NNρA
[9]   XX←-A+.×B←B+.×⍉((ρIPT),N)ρXX
[10]  N←(2,2↑ρS),0ρL←ρρρA← 2×A
[11] L1:→L1+T<L←L+ρρρS←S, 3 1 2 ⍉Nρ⍉(((B+XX×2○NN),XX×1○NN)⍸ND+A×2○NN←○FREQUENCY[L]
      )[OPT;]
[12]  →L2×ι∧/G[1 2 3],(0ρB←S[;;A-1]),(0ρXX←S[;;A←2×ιT]),0ρR←((2↑ρS),0)ρA←F←ND←XX←0
[13]  →L2×ιG[3],0ρR←R,(20×10⍟(0 0 ,G[1]×T)↓ND),(0 0 ,G[2]×T)↓ND←((B×B)+XX×XX)*
      0.5
[14]  R←R,(180÷○1)×(ρND)ρ(,1-2×XX<0)×¯2○1⌊¯1⌈,B÷ND
[15] L2:→L3×∧/ 1 1 =2↑ρR←R,((0 0 ,G[4]×T)↓B),(0 0 ,G[5]×T)↓XX
[16] L3:→0,R←,R
[17] ER:'NOT A PROPER NETWORK, SKL BRANCH, L M P R I, OR I/O  NOT SPECIFIED, OR IN
      CORRECT FREQUENCY VECTOR'
    ∇
```

```
      ∇LADFREQ[⎕]∇
    ∇ R←A LADFREQ NET;P;N;PP;S11;S12;S21;S22;G11;G12;G21;G22
[1]   →ER×⍳∨/(0=∨/A[5+⍳4]),(0=∨/(A←'LMPRI1234'∈A)[⍳5]),0=⍴NET
[2]   S11←S22←(PP←2,P←⍴FREQUENCY)⍴0
[3]   S12←S21←⍉(P,2)⍴ 1 0
[4]  L1:NET←(⍴N←((1↓NET)⍳987654321)↑NET)↓NET
[5]   →ER×⍳∨/(⍴N←'RI' FREQ N)≠ 2 2 ,P←2×P
[6]   G11←PP⍴P↑,N
[7]   G12←PP⍴P↑N←P↓,N
[8]   G21←PP⍴P↑N←P↓N
[9]   P←P÷2,N←0⍴G22←PP⍴P↓N
[10]  N←N÷PP⍴+/[1] N×N←(⍉(P,2)⍴ 1 0)-PP⍴(-/[1] G11×S22),-+/[1] G11×⊖S22
[11]  S21←PP⍴(-/[1] S21×N),+/[1] S21×⊖N
[12]  S12←PP⍴(-/[1] S12×G12),+/[1](N←S12)×⊖G12←PP⍴(-/[1] G12×N),+/[1] G12×⊖N
[13]  S22←G22+PP⍴(-/[1] S22×G12),+/[1] S22×⊖G12←PP⍴(-/[1] G21×G12),+/[1] G21×⊖G12
[14]  S21←PP⍴(-/[1] G21×G12),+/[1] G21×⊖G12←S21
[15]  S11←S11+PP⍴(-/[1] N×G11),+/[1] N×⊖G11←PP⍴(-/[1] G11×G12),+/[1] G11×⊖G12
[16]  →L1×⍳0≠⍴NET,G11←G12←G21←G22←N←⍳0
[17]  G11←S12[;⍳P×A[6]],S11[;⍳P×A[7]],S21[;⍳P×A[8]],S22[;⍳P×A[9]]
[18]  →L2×⍳0=∨/A[1 2 3],(,R←((N←+/A[6 7 8 9]),0)⍴0),S11←S12←S21←S22←⍳0
[19]  R←R,(20×10⊛(N,P×A[1])⍴G22),(N,P×A[2])⍴G22←(+/[1] G11×G11)*
      0.5
[20]  →L2×⍳0=A[3]
[21]  R←R,(180÷○1)×(N,P)⍴(1-2×,G11[2;]<0)×¯2○1⌊¯1⌈,G11[1;]÷G22
[22] L2:→L3×1=1↑⍴R←R,((N,P×A[4])⍴G11[1;]),(N,P×A[5])⍴G11[2;]
[23] L3:→0,R←,R
[24] ER:'NETWORKS NOT 2 INPUT 2 OUTPUT, OR LMPRI1234 NOT SPECIFIED'
    ∇
```

```
      ∇SENS[□]∇
    ∇ S←G SENS NET;L;A;B;N;NN;IO;XX;X;YY;Y;ND;ZZ;IPT;OPT;T;HD;R
[1]   →ER×⍳∨/(NET[1]≠987654321),(∧/G←~'RIMLT'∈G),(5<⍴G),0=T←⍴FREQUENCY←,FREQUENCY,
      0⍴L←(3⍴1000)⊤NET[2]
[2]   →ER×⍳∨/(∧/XX←200≥400|XX),(400<XX←1000|L[1]↓L[3]↑NET),(0=-/A[3 2]),0=L[1]-(A←
      (3⍴1000)⊤NET[3])[3]
[3]   NN←N,N←⍴ND←(N∈ND)/N←⍳⌈/ND←(IO←NET⌊1+A[2 3]]),, ¯1 0 ↓B←200|(3⍴1000)⊤L[1]↓L[
      3]↑NET
[4]   A←⍋(1000 1000 ⊥(XX←(3⍴1000)⊤XX/L[1]↓L[3]↑NET)[2 1 ;])+
      0.5×ZZ←((+/ZZ↑XX)⍴0),(+/(ZZ←-/L[2 1])↓XX←~XX)⍴1
[5]   R←0⍴(IO←ND⍳IO),(⍴ZZ←((⍴OPT),1)⍴ZZ[A]),(OPT←OPT[A]),(IPT←IPT[A]),(OPT←ND⍳,XX[
      2;]),IPT←ND⍳,XX[1;]
[6]   ND←(ND⍳,B[2;])+N×(ND⍳,B[1;])-⍴⍴NET←(L[3]↓NET)[, 2 0 ↓B]
[7]   B[L↑ND]←B[L↑ND]-(L←-/L[2 1])↑NET,B←,NN⍴((N,1)⍴1),NN⍴0
[8]   HD[L↓ND]←L↓NET,0⍴HD←,NN⍴0
[9]   X[(NET←IO[1]);]←Y[IO[2];]←⍴⍴⍴X←Y←(N,1)⍴0
[10]  A←B-(XX←HD+.×YY←-⌹B←NN⍴B)+.×HD←NN⍴HD
[11]  S←(NN[1 2],0)⍴N←N,L←B←ND←⍴⍴⍴NN←(N←⍴IPT),2,⍴⍴(⍴XX←XX+.×Y),(⍴HD←¯2×HD),⍴YY←(⌽
      YY+.×HD)+.×Y
[12] L1:R←R,(,(B←((Y+YY×2○ND),YY×1○ND)⌹⌽B)[NET;]),0⍴IO←(((X+XX×2○ND),XX×1○ND)⌹B←A+
      HD×2○ND←○FREQUENCY[L])[OPT;]
[13]  →L1×⍳T≥L←L+⍴⍴⍴S←S,NN⍴(N⍴-/B×IO),N⍴+/B×⌽IO←((~ZZ)+ZZ×2○ND),ZZ×-1○ND,0⍴B←(N⍴-/
      B×IO),N⍴+/IO×⌽B←B[IPT;]
[14]  →L2×∨/~G[3 4 5],0⍴S←⌽(0 0 ,G[1])↓(0 0 ,-G[2])↓A←⌽S
[15] L2:XX←+/A×((NN⍴X)÷HD),(NN⍴Y)÷HD←(NN←N[1],T,1)⍴(10*-70)⌈((X×X←R[¯1+2×⍳T])+Y×Y←
      R[2×⍳T])*0.5
[16]  →0,⍴S←⌽(⌽S),((0 0 ,G[3])↓NN⍴XX),((0 0 ,G[4])↓(NN⍴XX)÷HD),(0 0 ,G[5])↓HD
[17] ER:'IMPROPER NETWORK, NO * BRANCHES, SKL BRANCH, INPUT/OUTPUT NOT SPECIFIED,
      OR FREQUENCY NOT SPECIFIED'
    ∇
```

```
      ∇SENS2[□]∇
    ∇ S←SENS2 NET;L;A;B;N;NN;IO;XX;X;YY;Y;ND;ZZ;IPT;OPT;T;HD;R
[1]   →ER×⍳∨/(NET[1]≠987654321),0=T←⍴FREQUENCY←,FREQUENCY,0⍴L←(3⍴1000)⊤NET[
      2]
[2]   →ER×⍳∨/(∧/XX←200≥400|XX),(400<XX←1000|L[1]↓L[3]↑NFT),(0=-/A⌈3 2]),0=L[1]-(A←
      (3⍴1000)⊤NET[3])[3]
[3]   NN←N,N←⍴ND←(N∈ND)/N←⍳⌈/ND←(IO←NET[1+A[2 3]]),,  1 0 ↓B←200|(3⍴1000)⊤L[1]↓L[
      3]↑NET
[4]   A←⍋(1000 1000 ⊥(XX←(3⍴1000)⊤XX/L[1]↓L[3]↑NET)[2 1 ;])+
      0.5×ZZ←((+/ZZ↑XX)⍴0),(+/(ZZ←-/L[2 1])↓XX←~XX)⍴1
[5]   R←0⍴(IO←ND⍳IO),(⍴ZZ←((⍴OPT),1)⍴ZZ[A]),(OPT←OPT[A]),(IPT←IPT[A]),(OPT←ND⍳,XX[
      2,⌋),IPT←ND⍳,XX[1;]
[6]   ND←(ND⍳,B[2;])+N×(ND⍳,B[1;])-⍴⍴NET←(L[3]↓NFT)[, 2 0 ↓B]
[7]   B[L↑ND]←B[L↑ND]-(L←-/L[2 1])↑NET,B←,NN⍴((N,1)⍴1),NN⍴0
[8]   HD[L↓ND]←L↓NET,0⍴HD←,NN⍴0
[9]   X[(NET←IO[1]);]←Y[IO[2];]←⍴⍴⍴X←Y←(N,1)⍴0
[10]  A←B-(XX←HD+.×YY←-⌸B←NN⍴B)+.×HD←NN⍴HD
[11]  S← 2 0 ⍴N←N,L←B←ND←⍴⍴⍴NN←(N←⍴IPT),2,⍴⍴(⍴XX←XX+.×X),(⍴HD←¯2×HD),⍴YY←(⌽YY+.×HD
      )+.×Y
[12]  L1:R←(,(B←((Y+YY×2○ND),YY×1○ND)⌸⌽B)[NET;]),0⍴IO←(((X+XX×2○ND),XX×1○ND)⌸B←A+HD
      ×2○ND←○FREQUENCY[L])[OPT;]
[13]  B←(N⍴-/B×IO),N⍴+/B×⍵IO←((~ZZ)+ZZ×2○ND),ZZ×-1○ND,0⍴B←(N⍴-/B×IO),N⍴+/IO×⍵B←B[
      IPT;]
[14]  →L1×T≥L←L+⍴⍴⍴S←S, 2 1 ⍴((+/B×B←+/B×NN[1 2]⍴R)÷IO),(IO←+/R×R)*
      0.5
[15] ER:'IMPROPER NETWORK, NO * BRANCHES, SKL BRANCH, INPUT/OUTPUT NOT SPECIFIED,
      OR FREQUENCY NOT SPECIFIED'
    ∇
```

```
      ∇PLOT[⎕]∇
    ∇ A PLOT B;C;D;F;G;H;I;J;L;T;Y;HZ;NB;VT;PT;ST;ISV;U
[1]   PC←'*∘○□Δ∇'
[2]   HS←0
[3]   ST← 1 2 5
[4]   SM← 5 10
[5]   →((0=×/(2ρA),ρB), 3 2 1 <ρρB)/0,PRKERR,PLOTL2,PLOTL3
[6]   →PLOTL3×ρρρB←⍉(2,D)ρ(⍳D←ρ,B),B
[7]  PLOTL2:B←B[1;;]
[8]  PLOTL3:Y←1+⍳(ρB)[2]-1
[9]   C←((⌈/⌈/B[;Y])-⌊/⌊/B[;Y]),(⌈/B[;1])-⌊/B[;1]
[10]  F←|(2ρA)÷C+(C=0)×B[1; 2 1]+B[1; 2 1]=0
[11]  F←(ST[+/(⌊0.0001+F×10*-G)∘.≥ST])×10*G←⌊10⊛F
[12]  C←(SM÷F)×⌊((⌊/⌊/B[;Y]),⌊/B[;1])×F÷SM←16⌊SM⌈ 1 4
[13]  B[;1]←⌊0.5+F[2]×B[;1]-G[2]
[14]  B[;Y]←⌊0.5+F[1]×B[;Y]-G[1]
[15]  H←SM×⌈((⌈/⌈/B[;Y]),⌈/B[;1])÷SM
[16]  NB←G[1]+(SM[1]÷F[1])×0,⍳H[1]÷SM[1]
[17]  HZ←G[2]+(SM[2]÷F[2])×0,⍳H[2]÷SM[2]
[18]  0ρST←6ρ~ISV←1≠U←9
[19] PLOTL4:VT←∨/0>NB←NB×10*U-ST[6-ISV]←I←1+⌈/⌊10⊛|(NB≠0)/NB
[20]  PT←⌊1+10|PT-1|PT←1E¯5+(|NB)∘.÷10*¯1+⍳U
[21]  L←U+1-(⍳((C←ρNB)ρ1)∧.=PT)⍳0
[22]  →((U>T←VT+⌈/I,(L+L≠I),(I≤0)×2+L-I),ST[2-ISV]←ST[2-ISV]∨L≥U-VT+L>I)/
      3 2 +I26
[23]  →(¯1+I26)×ρρρST[4-ISV]←I←1
[24]  →PLOTL4×ρρNB←SM[1+~ISV]×¯1+⍳C
[25]  PT←(-VT+0⌈1-I)⍳PT
[26]  PT←(,PT)×J←,⍉(⍳ρPT)ρ(,⍉(1≠PT)∨.∧(⍳U)∘.≤⍳VT⌈I-1),(C×U+1-I←VT+I⌈I≤0)ρ1
[27]  →(2+I26)×⍳~VT
[28]  PT[(U-+/(C,U)ρJ)+U×¯1+⍳C]←11×NB<0
[29]  PT←(~(⍳U+J)∈(I+J),⍳¯1+J←U-T)\(1 0 +C,U)ρPT,Uρ0
```

```
      ∇PLOT[⎕30]∇
[30]  PT[⍳C;I+J]←12
[31]  PT←' 0123456789¯.'[1+PT[;⍳U-1]]
[32]  →PLOTL13×⍳~ISV
[33]  L←1,H[2]⍴0×C←H[1]
[34] PLOTL8:L←(L×HS×C≠0)⌈1,H[2]⍴0
[35]  L[1+(D≠0)/B[;1]]←(D≠0)/D←(C=B[;Y])⌈.×Y
[36]  →(C≠0)/PLOTL11
[37]  L←L⌈0=(SM[2]÷2)|0,⍳H[2]
[38] PLOTL11:PT[((⍴PT)[1],1+C÷SM[1])[1+0=SM[1]|C];],(' |',(⍴Y)⍴PC)[1+L]
[39]  →(0≤C←C-1)⍴PLOTL8
[40]  →(U=U←SM[2]-~ISV←~⍴⍴NB←,HZ)⍴PLOTL4
[41] PLOTL13:(SM[2]-9)⍎(,(0 0 ,(U-1)⍴1)\PT),' '
[42]  →(ST[1 3 2 4],1)/ 1 3 5 7 10 +⍳26
[43]  ('ORIGIN AND SCALF FACTOR FOR ORDINATE: ';G[1],÷F[1])
[44]  →(2+⍳26)×⍳0=ST[3]
[45]  ('SCALE FACTOR FOR ORDINATE: ';10*ST[5]-1)
[46]  →(2+⍳26)×⍳0=ST[2]
[47]  ('ORIGIN AND SCALE FACTOR FOR ABSCISSA: ';G[2],÷F[2])
[48]  →0×⍳0=ST[4]
[49]  ('SCALE FACTOR FOR ABSCISSA: ';10*ST[6]-1)
[50]  →0
[51] PRKERR:'THE RIGHT ARGUMENT OF  PLOT  MUST HAVE RANK ≤ 3.'
    G∇
```

```
      ∇AND[⎕]∇
    ∇ L←A AND B;C;D
[1]   →(((2<ρρA)∨3<ρρB),0≠ρρB)/ 17 3
[2]   B←,B
[3]   →(((3=ρρB)∧1≠1ρρB),2=ρρA)/ 17 7
[4]   A←,A
[5]   →(∧/((ρA)≠1,D),1≠D←1ρ¯2ϕρB)/16
[6]   A←(((D×ρA)⌊D⌈ρA),1)ρA
[7]   →(1≠ρρB)/9
[8]   B←(((ρB)⌈(1=ρB)×1ρρA),1)ρB
[9]   →((∧/D≠1,1ρρA),1≠D←1ρ¯2ϕρB)/ 16 11
[10]  B←(((3=ρρB)ρ1),(1ρρA),1ρϕρB)ρB
[11]  →(3=ρρB)/14
[12]  L←(((C←1ρϕρA)ρ0),(1ρϕρB)ρ1)\B
[13]  →0×ρρL[;⍳C]←A
[14]  L←(1,((C←1ρϕρA)ρ0),(¯1+1ρϕρB)ρ1)\B
[15]  →0×ρρL[;1+⍳C]←A
[16]  →0=ρ⎕←'ARGUMENTS OF  AND  ARE NOT CONFORMABLE.'
[17]  'AN ARGUMENT OF  AND  IS OF IMPROPER RANK.'
    ∇


      ∇VS[⎕]∇
    ∇ M←A VS B;C;D
[1]   →(((ρρB←,B)<ρρB), 2 1 0 <ρρA)/ 8 8 4 3
[2]   A←((ρB),1)ρA
[3]   A←((×/ρA),1)ρA
[4]   →(∧/(ρB)≠1,1ρρA)/9
[5]   M←(0,(1ρϕρA)ρ1)\A
[6]   M[;⍳1]←B
[7]   →0×ρρM←(1,ρM)ρM
[8]   →0=ρ⎕←'AN ARGUMENT OF  VS  IS OF IMPROPER RANK.'
[9]   'ARGUMENTS OF  VS  ARE NOT CONFORMABLE.'
    ∇
```

REFERENCES

[1] J. F. Kaiser, "Digital Filters", Ch. 7 in Systems Analysis by Digital Computer, F. F. Kuo and J. F. Kaiser, Eds. New York, Wiley, 1966.

[2] B. Gold and C. M. Rader, Digital Processing of Signals, McGraw-Hill, New York, 1969.

[3] A. V. Oppenheim and R. W. Schafer, Digital Signal Processing, Prentice Hall (in press).

[4] Digital Signal Processing (collection of papers) L. R. Rabiner and C. M. Rader, Eds. IEEE Press, New York, 1972.

[5] S. J. Mason, "Feedback theory- some properties of signal flow graphs," Proc. IRE, Vol. 41, pp. 1144-1156, September 1953.

[6] S. J. Mason, "Feedback theory- further properties of signal flow graphs," Proc. IRE, Vol. 44, pp. 920-926, July 1956.

[7] R. E. Crochiere, "Some network properties of digital filters," Quarterly Progress Report No. 107, Research Laboratory of Electronics, M.I.T., pp. 103-112, October 15, 1972, see also, Proc. 1973 IEEE Int. Symp. on circuit theory, pp. 146-148.

[8] G. E. Forsythe and C. B. Moler, Computer Solution of Linear Algebraic Systems, Prentice-Hall, 1967.

[9] R. W. Hamming, Numerical Methods for Scientists and Engineers, McGraw Hill, 1973.

[10] Y. Chow and E. Cassignol, Linear Signal-Flow Graphs and Applications, John Wiley & Sons, New York, 1962.

[11] V. N. Faddeeva, Computational Methods of Linear Algebra, Dover Publications, New York, 1959.

[12] P. M. DeRusso, R. J. Roy, and C. M. Close, State Variables for Engineers, John Wiley & Sons, New York, 1965.

[13] D. P. Lindorff, Theory of Sampled-Data Control Systems, New York, Wiley, 1965.

[14] R. W. Brockett, Finite Dimensional Linear Systems, John Wiley & Sons, New York, 1970.

[15] A. Fettweis, "Digital filter structures related to classical filter networks," Arch. Elekt. Übertrag., vol. 25, pp. 79ff, February 1971.

[16] A. Sedlmeyer and A. Fettweis, "Realization of digital filters with true ladder configuration," Proc. 1973 IEEE Int. Symp. on Circuit Theory, pp. 149-152, see also, Int. J. Circuit Theory and Appl., vol. 1, No. 1.

[17] R. E. Crochiere, "Digital ladder structures and coefficient sensitivity," IEEE Trans Audio Electroacoust., vol. AU-20, No. 4, pp. 240-246, October 1972.

[18] S. K. Mitra and R. J. Sherwood, "Digital ladder networks," IEEE Trans. Audio Electroacoust., vol. AU-21, pp. 30-36, February 1973.

[19] P. Penfield, Jr., R. Spence, and S. Duinker, Tellegen's Theorem and Electric Networks, MIT Press, Cambridge, Mass., 1970.

[20] R. E. Seviora and M. Sablatash, "A Tellegen's theorem for digital filters," IEEE Trans. on Circuit Theory, vol. CT-18, pp. 201-203, January 1971.

[21] A. Fettweis, "A general theorem for signal-flow networks, with applications," Arch. Elekt. Übertrag., vol. 25, pp. 557-561, December 1971, see also, Digest of Technical Papers, London, 1971 IEEE Int. Symp. on Electrical Network Theory, pp. 3-4.

[22] A. Y. Lee, "Topological properties of the summing and branching matrices of signal-flow graphs and the application to sensitivity analysis," Proc. of the 6th Asilomar Conf. on Circuits and Systems, November 1972.

[23] A. Y. Lee, "Computer-aided equation formulation and sensitivity analysis for discrete systems," Proc. of the 16th Midwest Symp. on Circuit Theory, April 1973.

[24] L. B. Jackson, "On the interaction of roundoff noise and dynamic range in digital filters," Bell Syst. Tech. J., pp. 159-184, February 1970.

[25] J. Truxal, Automatic Feedback Control Synthesis, McGraw Hill, New York, 1955.

[26] I. M. Horowitz, Synthesis of Feedback Systems, Academic Press, New York, 1963.

[27] R. E. Crochiere, "Computational methods for sensitivity analysis of digital filters," Quarterly Progress Report No. 109, Research Laboratory of Electronics, M.I.T., pp. 113-123, April 15, 1973.

[28] M. L. Blostein, "Some bounds on sensitivity in RLC networks," Proc. First Allerton Conf. on Circuits and Systems, October 1963.

[29] C. Belove, "Sensitivity sums of homogeneous functions," IEEE Trans. CT-11, pp. 171, March 1964.

[30] K. Géher, Theory of Network Tolerances, Akadémiai Kiadó, Budapest, 1971.

[31] S. K. Mitra and R. J. Sherwood, "Canonic realizations of digital filters using the continued fraction expansion," IEEE Trans. Audio Electroacoust., vol. AU-20, No. 3, pp. 185-194, August 1972.

[32] L. B. Jackson, "Roundoff-noise analysis for fixed-point digital filters realized in cascade or parallel form," IEEE Trans. Audio Electroacoust., vol. AU-18, pp. 107-122, June 1970.

[33] S. R. Parker and S. Hess, "Canonic realizations of second-order digital filters due to finite precision arithmetic," IEEE Trans. Circuit Theory, vol. CT-17, pp. 410-413, July 1970.

[34] E. Avenhaus, "A proposal to find suitable canonical structures for the implementation of digital filters with small coefficient word length," Nachrichtentechn. Z., vol. 25, pp. 377-382, August 1972.

[35] A. H. Gray and J. D. Markel, "Digital lattice and ladder filter synthesis," IEEE Trans. Audio Electroacoust., vol. AU-21, pp. 491-500, December 1973.

[36] R. E. Crochiere, "On the location of zeros and a reduction in the number of adds in a digital ladder structure," IEEE Trans. Audio Electroacoust., vol. AU-21, pp. 551-552, December 1973.

[37] A. Chu and R. E. Crochiere, "Comments and experimental results on optimal digital ladder structures," IEEE Trans. Audio Electroacoust., vol. AU-20, pp. 317-318, October 1972.

[38] R. E. Crochiere, "A user's guide to the computer-aided digital network analysis package (CADNAP)," M.I.T., 1974.

[39] W. J. McCalla and D. O. Pederson, "Elements of computer-aided circuit analysis," IEEE Trans. Circuit Theory, vol. CT-18, pp. 14-26, January 1971.

[40] F. H. Branin, "Computer methods of network analysis," Proc. IEEE, vol. 55, No. 11, pp. 1787-1801, November 1967.

[41] Proceedings of the IEEE, special issue on computer-aided design, vol. 55, No. 11, November 1967.

[42] IEEE Trans. on Circuit Theory, special issue on computer-aided circuit design, vol. CT-18, No. 1, January 1971.

[43] J. G. F. Francis, "The Q-R transformation, part I," Computer J., vol. 4, pp. 265-271, October 1961; "The Q-R transformation, part II," Computer J., vol. 4, pp. 332-345, January 1962.

[44] J. H. Wilkinson, The Algebraic Eigenvalue Problem. London, England: Oxford University Press, 1965, pp. 434-435.

[45] R. W. Brockett, "Poles, zeros, and feedback: state space interpretation," IEEE Trans. Automatic Control, vol. AC-10, pp. 129-135, April 1965.

[46] I. W. Sandberg and H. C. So, "A two-sets-of-eigenvalues approach to the computer analysis of linear systems," IEEE Trans. Circuit Theory, vol. CT-16, pp. 509-517, November 1969.

[47] A. J. Goldstein and F. F. Kuo, "Multiparameter sensitivity," IEEE Trans. Circuit Theory (Correspondence), vol. CT-8, pp. 177-178, June 1961.

[48] S. R. Parker, "Sensitivity: Old questions, some new answers," IEEE Trans. Circuit Theory, vol. CT-18, no. 1, pp. 27-34, January 1971.

[49] R. E. Crochiere, "A new statistical approach to the coefficient word length problem for digital filters," presented at the Arden House Workshop on Digital Signal Processing, Harriman, New York, January 14-17, 1974; To be presented at the IEEE Int. Symp. on Circuit Theory, San Francisco, California, April 21-24, 1974; Also submitted for publication in IEEE Trans. Circuit Theory.

[50] J. B. Knowles and E. M. Olcayto, "Coefficient accuracy and digital filter response," IEEE Trans. Circuit Theory, vol. CT-15, pp. 31-41, March 1968.

[51] S. W. Director and R. A. Rohrer, "The Generalized Adjoint Network and Network Sensitivities," IEEE Trans. Circuit Theory, vol. CT-16, pp. 318-323, August 1969. Also

"Automated Network Design - The Frequency Domain Case", pp. 330-337.

[52] A. V. Oppenheim and D. H. Johnson, "Discrete Representation of Signals," IEEE Proc., vol. 60, no. 6, pp. 681-691, June 1972.

[53] J. F. Pinel and M. L. Blostein, "Computer techniques for the frequency analysis of linear electrical networks," IEEE Proc., vol. 55, no. 11, pp. 1810-1819, Nov. 1967.

[54] L. Gilman and A. J. Rose, APL 360 an Interactive Approach (John Wiley and Sons, Inc., New York, 1970).

[55] APL\360 Primer and Users Manual, IBM Corp.

[56] R. J. Deaton, "An Aid for Teaching Digital Signal Processing" S.M. Thesis, Department of Electrical Engineering, M.I.T., June 1970.

[57] E. Avenhaus, "On the design of digital filters with coefficients of limited word length," IEEE Trans. Audio Electroacoust., vol. AU-20, pp. 206-212, August 1972.

[58] A. Fettweis, "Pseudopassivity, sensitivity and stability of wave digital filters," IEEE Trans. Circuit Theory, vol. CT-19, pp. 668-673, November 1972.

[59] D. C. Youla, "A tutorial exposition of some key network-theoretic ideas underlying classical insertion-loss filter design," Proc. IEEE, vol. 59, pp. 760-799, May 1971.

[60] L. R. Rabiner, J. F. Kaiser, O. Herrmann, and M. Dolan, "Some comparisons between FIR and IIR filters," B.S.T.J., February 1974, to be published.

[61] A. G. Constantinides, "Spectral transformations for digital filters," Proc. IEEE, vol. 117, pp. 1585-1590, August 1970.

[62] C. Hastings, Jr., Approximations for Digital Computers (Princeton University Press, Princeton, New Jersey, 1955).

[63] P. F. Byrd and M. D. Friedman, Handbook of Elliptic Integr ls for Engineers and Physicists (Lange, Maxwell, and Springer, Berlin, 1954).

[64] K. Renner and S. C. Gupta, "On the design of wave digital filters with low sensitivity properties," IEEE Trans. Circuit Theory, vol. CT-20, pp. 555-567, September 1973.

[65] H. Wakita, "Direct estimation of the vocal tract shape by inverse filtering of acoustic speech waveforms," IEEE Trans. Audio Electroacoust., vol. AU-21, pp. 417-427, October 1973.

[66] A Fettweis, "Some principles of designing filters imitating classical filter structures," IEEE Trans. Circuit Theory, vol. CT-18, pp. 314-316, March, 1971.

[67] R. Saal, Der Entwurf von Filtern mit Hilfe des Kataloges normierter Tiefpässe, Backnang/Württ, Western Germany, Telefunken G. M. B. H. 1961.

[68] D. S. K. Chan and L. R. Rabiner, "Analysis of quantization errors in the direct form for finite impulse response digital filters," IEEE Trans. Audio Electroacoust., vol. AU-21, pp. 354-366, August, 1973.

[69] L. B. Jackson, J. F. Kaiser, and H. S. McDonald, "An approach to the implementation of digital filters," IEEE Trans. Audio Electroacoust., vol. AU-16, pp. 413-421, September 1968.

[70] A. V. Oppenheim and C. J. Weinstein, "Effects of finite register length in digital filtering and the fast fourier transform," Proc. IEEE, vol. 60, No. 8, pp. 957-976, August 1972.

[71] H. W. Schüssler, Digitale Systeme zur Signalverarbeitung, Springer-Verlag, Berlin, West Germany 1973.

[72] B. C. Kuo, Discrete-Data Control Systems, Prentice-Hall, Englewood Cliffs, N.J., 1970.

[73] Literature in Digital Signal Processing, H. D. Helms and L. R. Rabiner, Eds., IEEE Press, New York, 1972.

[74] Papers on Digital Signal Processing, A. V. Oppenheim, Ed., M.I.T. Press, Cambridge, Mass., 1969.

BIOGRAPHICAL NOTE

Ronald E. Crochiere was born in Wausau, Wisconsin on September 28, 1945. He received the B.S. degree in electrical engineering from the Milwaukee School of Engineering, Milwaukee, Wisconsin, in 1967 and the S.M. and E.E. degrees in electrical engineering in 1968 and 1972 respectively from the Massachusetts Institute of Technology, Cambridge, Massachusetts.

From 1966 to 1967 he was employed, part time, with the physics department of the Milwaukee School of Engineering where he was involved in the design and construction of electronic equipment for scintillation spectrometry, pulse height analysis, and xray diffraction. From 1968 to 1970 he was employed with Raytheon Company, Wayland, Massachusetts where he was involved in the development of microwave sideband generators and high power microwave phase shifters. In 1970 he returned to the Massachusetts Institute of Technology to pursue further graduate studies. He joined the Research Laboratory of Electronics and became involved in research in the area of digital signal processing and computer-aided design.

Ronald Crochiere is a member of Kappa Eta Kappa, Sigma Xi and the Institute of Electrical and Electronics Engineers.